FIG. 1A

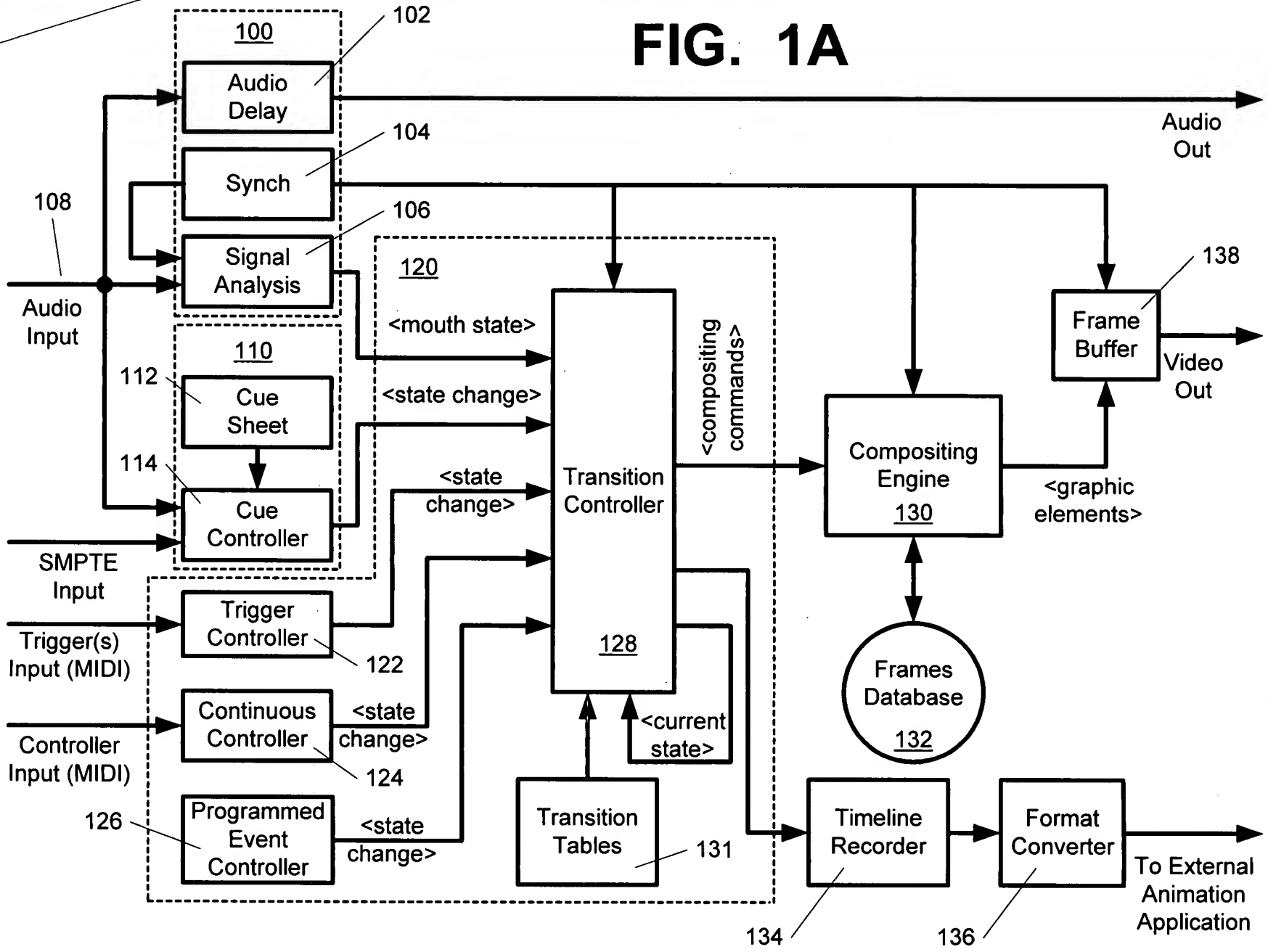

100
102
Audio Delay
Audio Out
104
Synch
108
106
Signal Analysis
120
Audio Input
<mouth state>
110
112
Cue Sheet
<state change>
114
Cue Controller
<state change>
Transition Controller
<compositing commands>
Compositing Engine
130
138
Frame Buffer
Video Out
<graphic elements>
SMPTE Input
Trigger Controller
122
Trigger(s) Input (MIDI)
128
Frames Database
132
Continuous Controller
<state change>
124
Controller Input (MIDI)
<current state>
126
Programmed Event Controller
<state change>
Transition Tables
131
Timeline Recorder
Format Converter
To External Animation Application
134
136

FIG. 1B

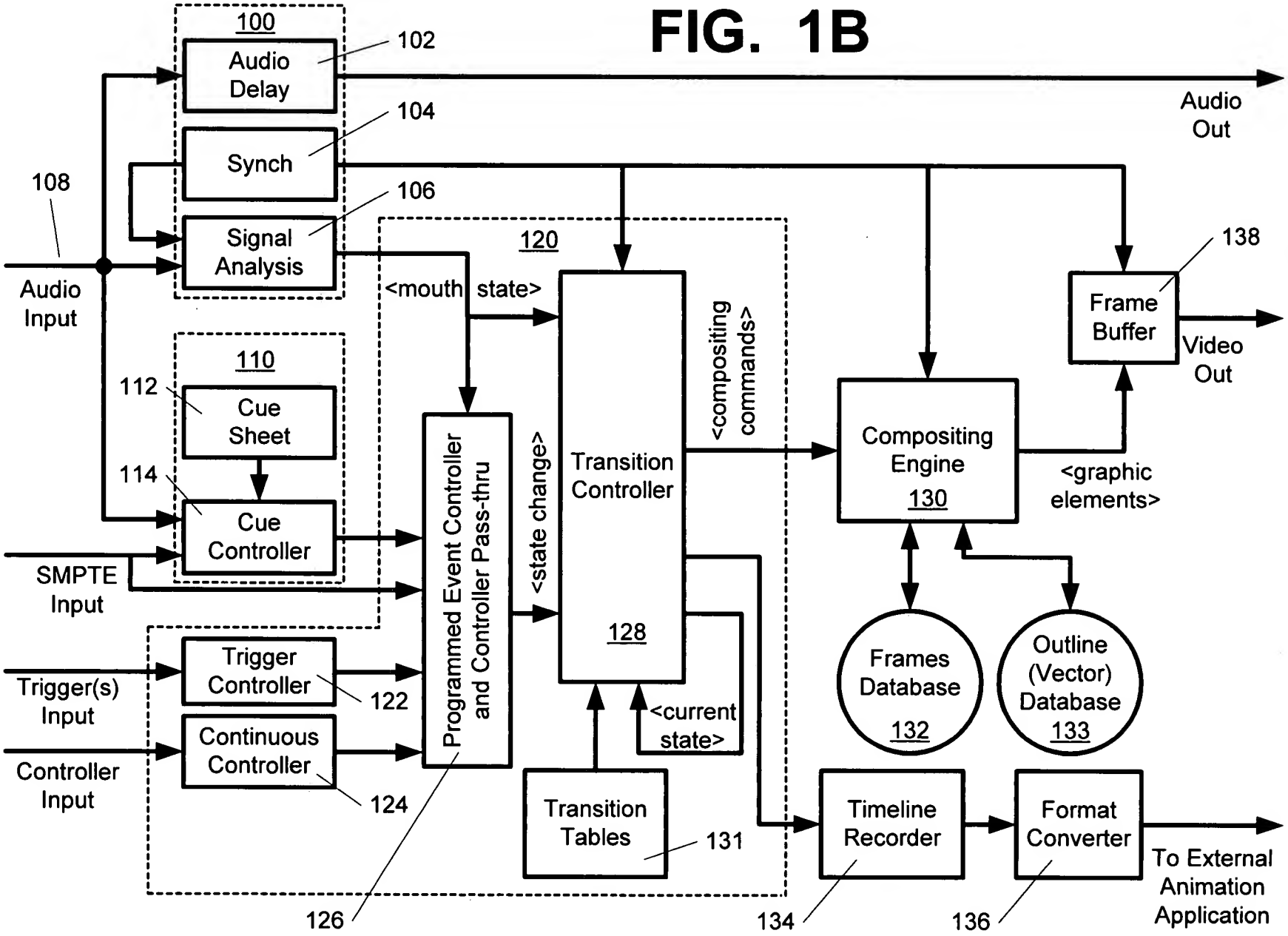

100
Audio Delay
102
Synch
104
Signal Analysis
106
108
Audio Input
Audio Out
120
<mouth state>
110
112
Cue Sheet
114
Cue Controller
SMPTE Input
Trigger Controller
122
Trigger(s) Input
Continuous Controller
124
Controller Input
Programmed Event Controller and Controller Pass-thru
<state change>
Transition Controller
128
<compositing commands>
Compositing Engine
130
Frame Buffer
138
Video Out
<graphic elements>
Frames Database
132
Outline (Vector) Database
133
<current state>
Transition Tables
131
Timeline Recorder
134
Format Converter
136
To External Animation Application
126

FIG. 2A

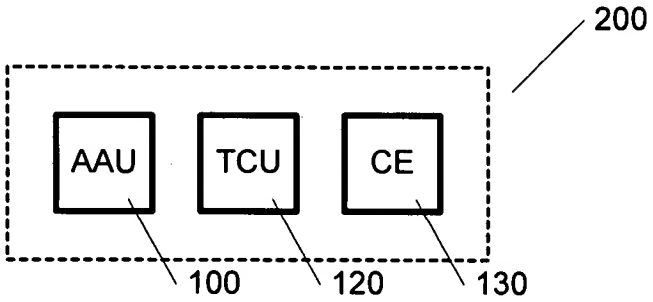
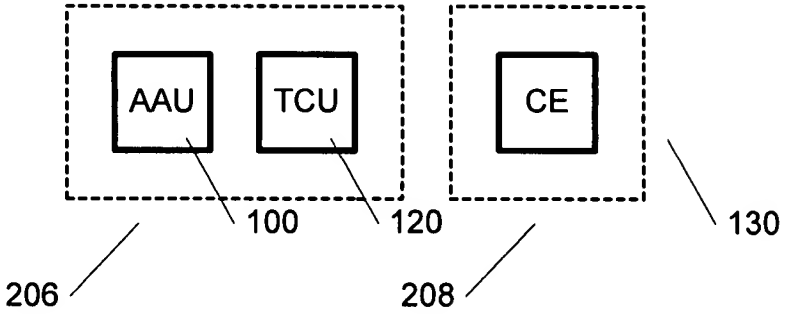


FIG. 2C



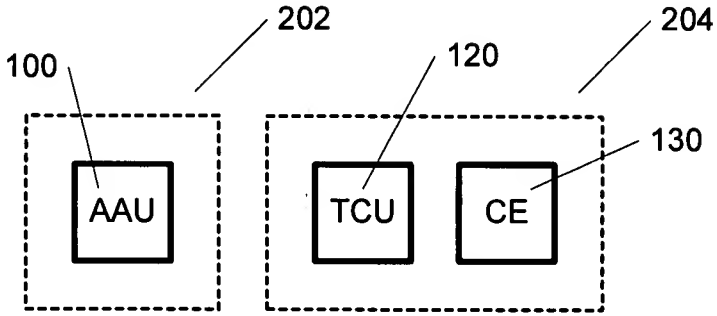


FIG. 2B

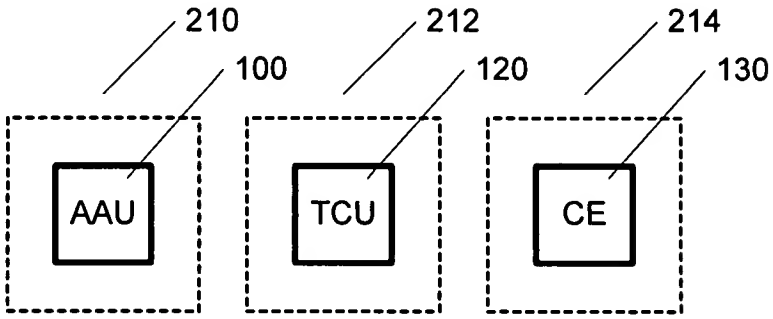


FIG. 2D

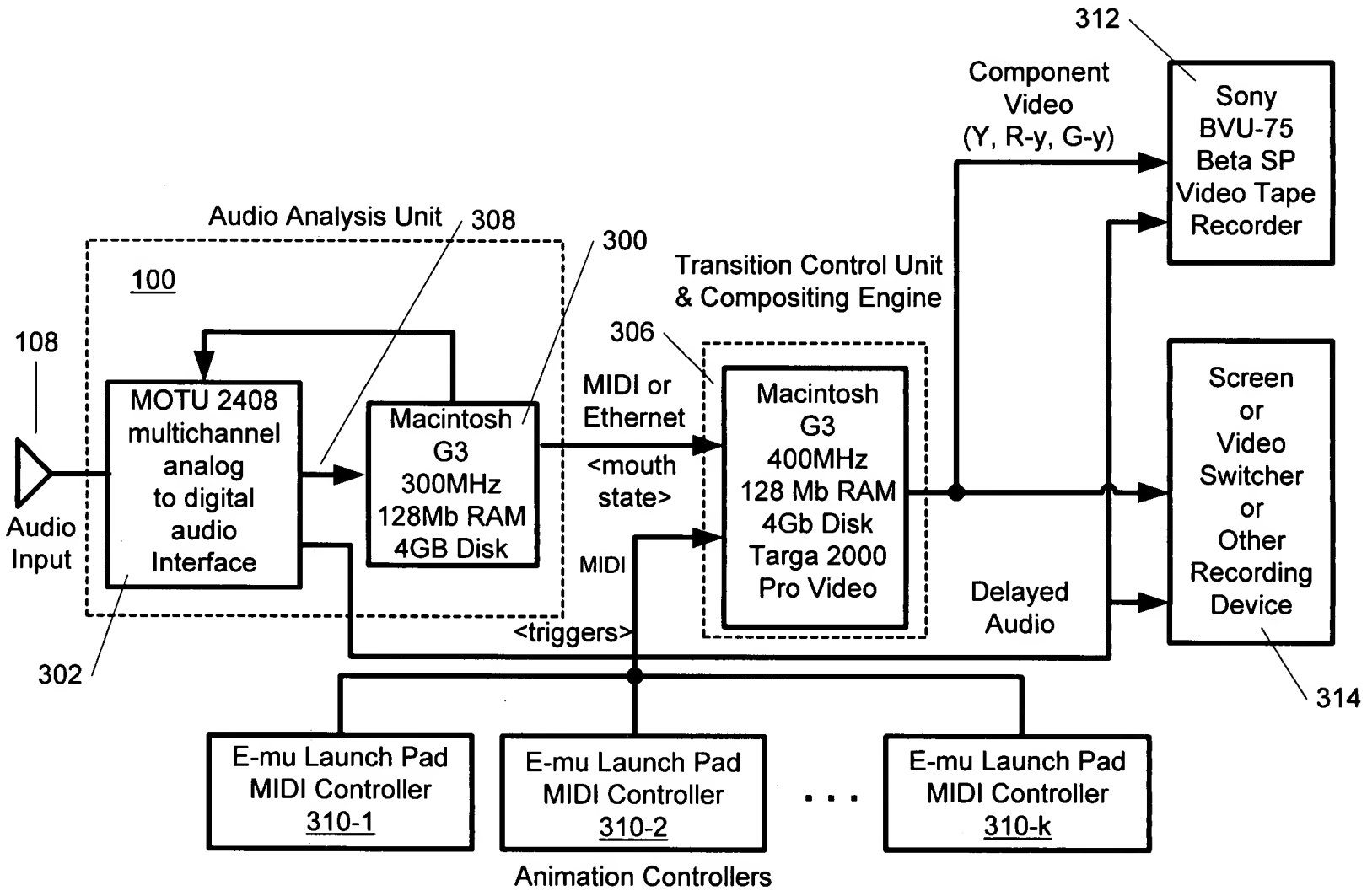

312
Component Video (Y, R-y, G-y)
Sony BVU-75 Beta SP Video Tape Recorder
Audio Analysis Unit
308
300
100
Transition Control Unit & Compositing Engine
306
108
MOTU 2408 multichannel analog to digital audio Interface
Macintosh G3 300MHz 128Mb RAM 4GB Disk
MIDI or Ethernet
<mouth state>
Macintosh G3 400MHz 128 Mb RAM 4Gb Disk Targa 2000 Pro Video
Screen or Video Switcher or Other Recording Device
Audio Input
MIDI
Delayed Audio
<triggers>
302
314
E-mu Launch Pad MIDI Controller 310-1
E-mu Launch Pad MIDI Controller 310-2
E-mu Launch Pad MIDI Controller 310-k
Animation Controllers

FIG. 3

FIG. 4

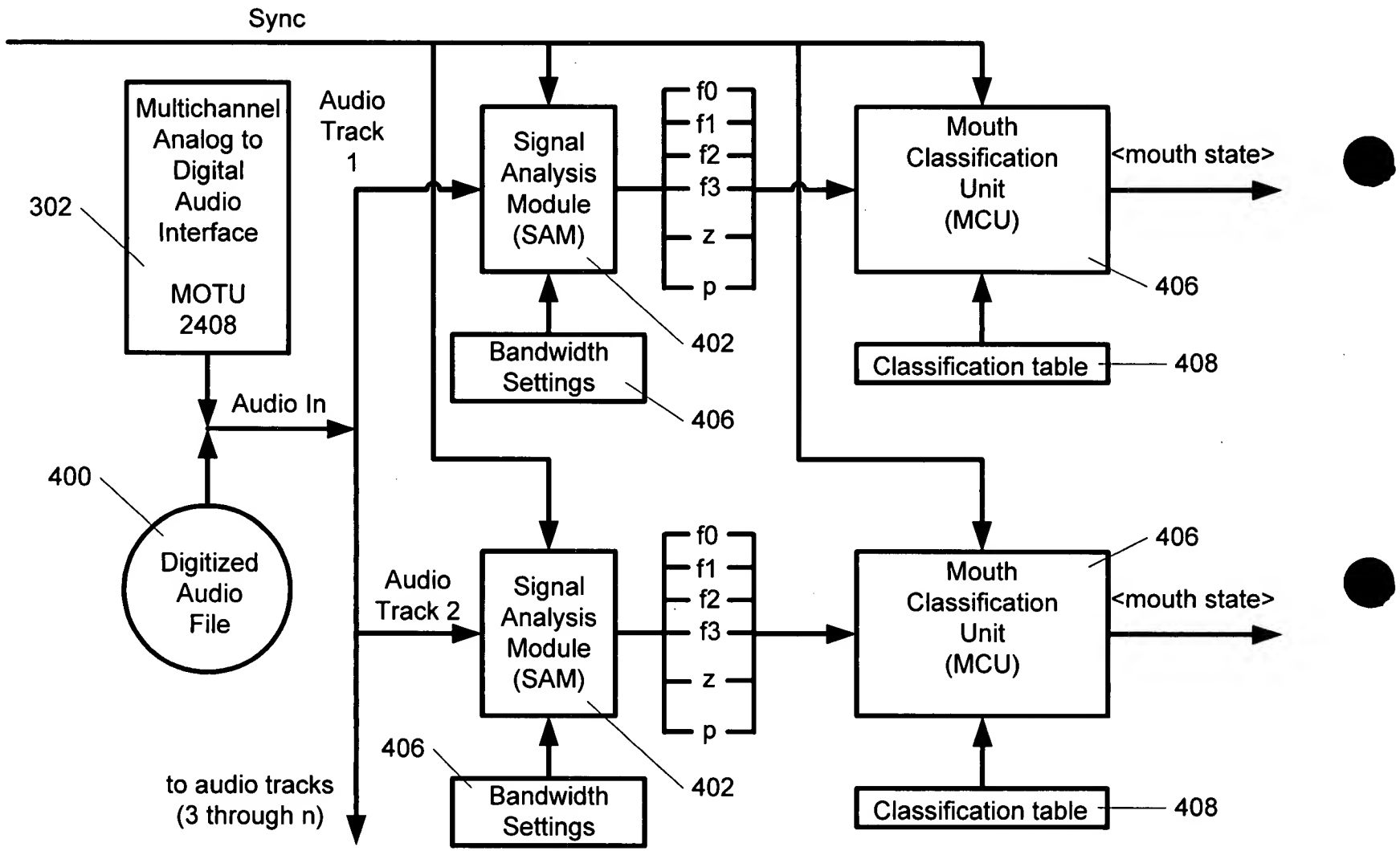
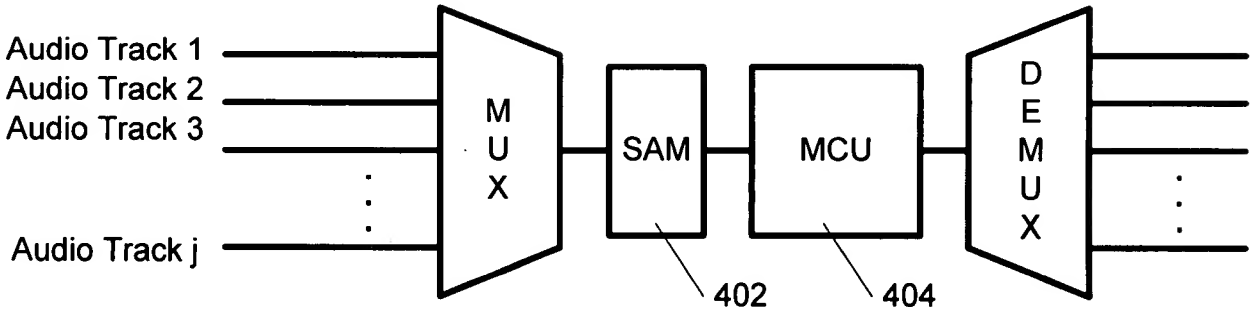

Sync
Multichannel Analog to Digital Audio Interface
MOTU 2408
302
Audio Track 1
Audio In
400
Digitized Audio File
Signal Analysis Module (SAM)
f0
f1
f2
f3
z
p
402
Bandwidth Settings
406
Mouth Classification Unit (MCU)
<mouth state>
406
Classification table
408
Audio Track 2
Signal Analysis Module (SAM)
f0
f1
f2
f3
z
p
402
406
Bandwidth Settings
Mouth Classification Unit (MCU)
406
<mouth state>
Classification table
408
to audio tracks (3 through n)

FIG. 5A


Audio Track 1
Audio Track 2
Audio Track 3
Audio Track j
MUX
SAM
MCU
DEMUX
402
404

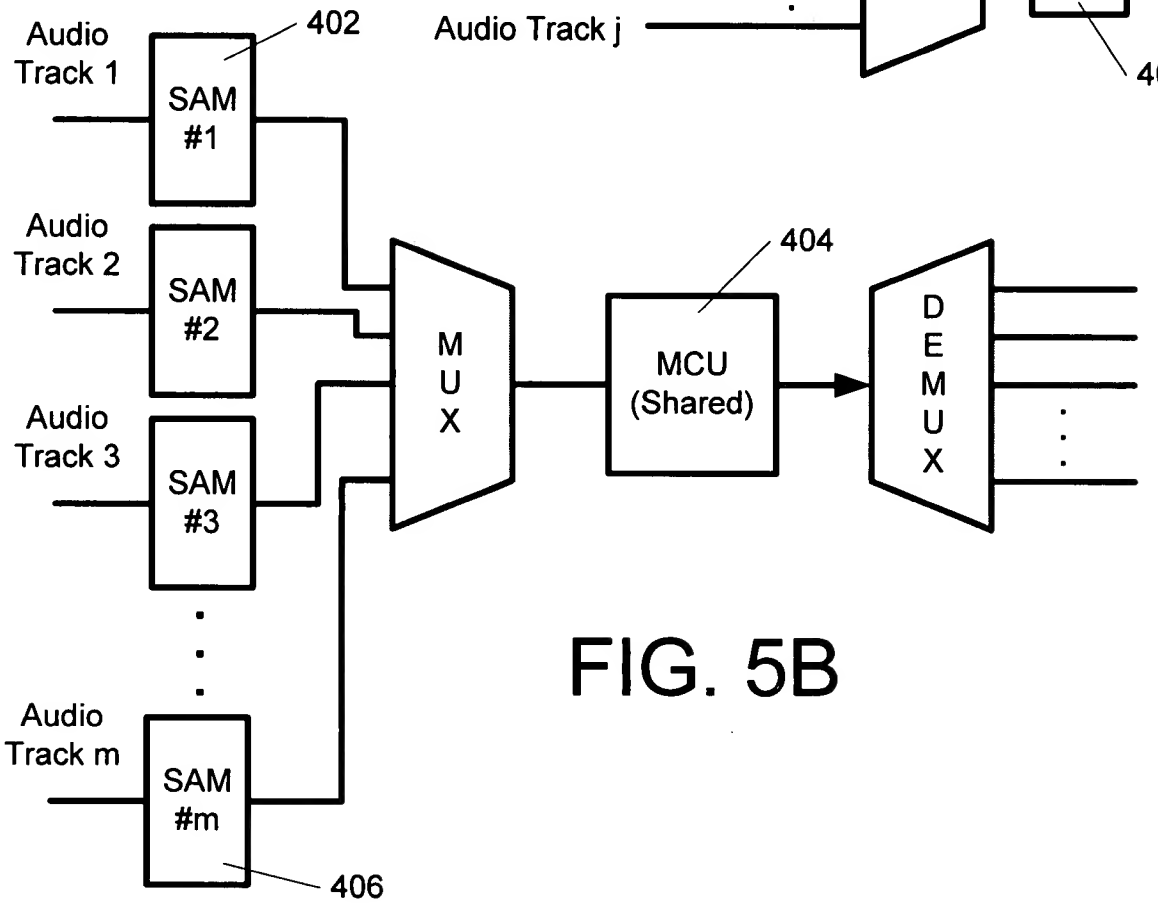

Audio Track 1
Audio Track 2
Audio Track 3
Audio Track m
SAM #1
SAM #2
SAM #3
SAM #m
402
MUX
MCU (Shared)
404
DEMUX
406

FIG. 5B

FIG. 6

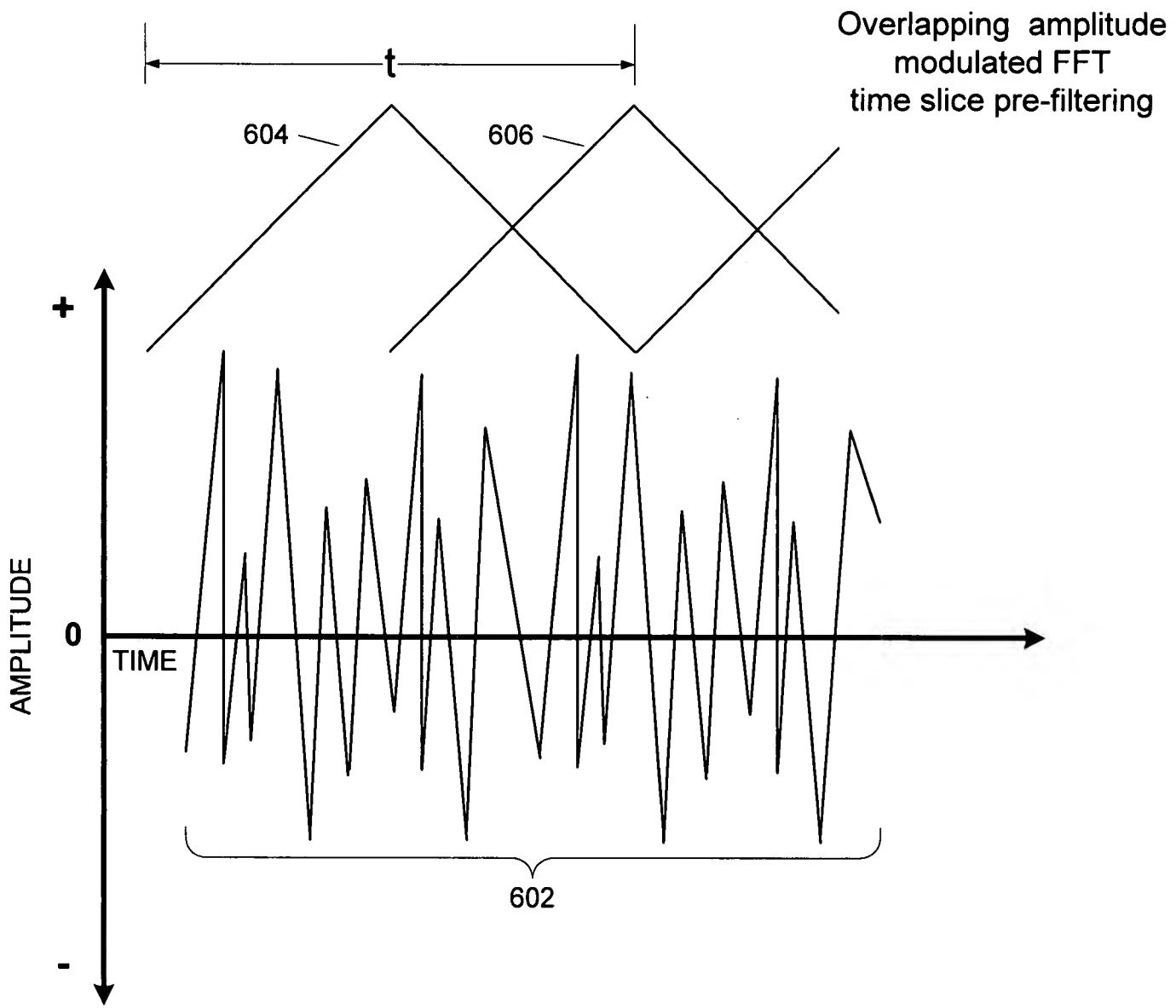
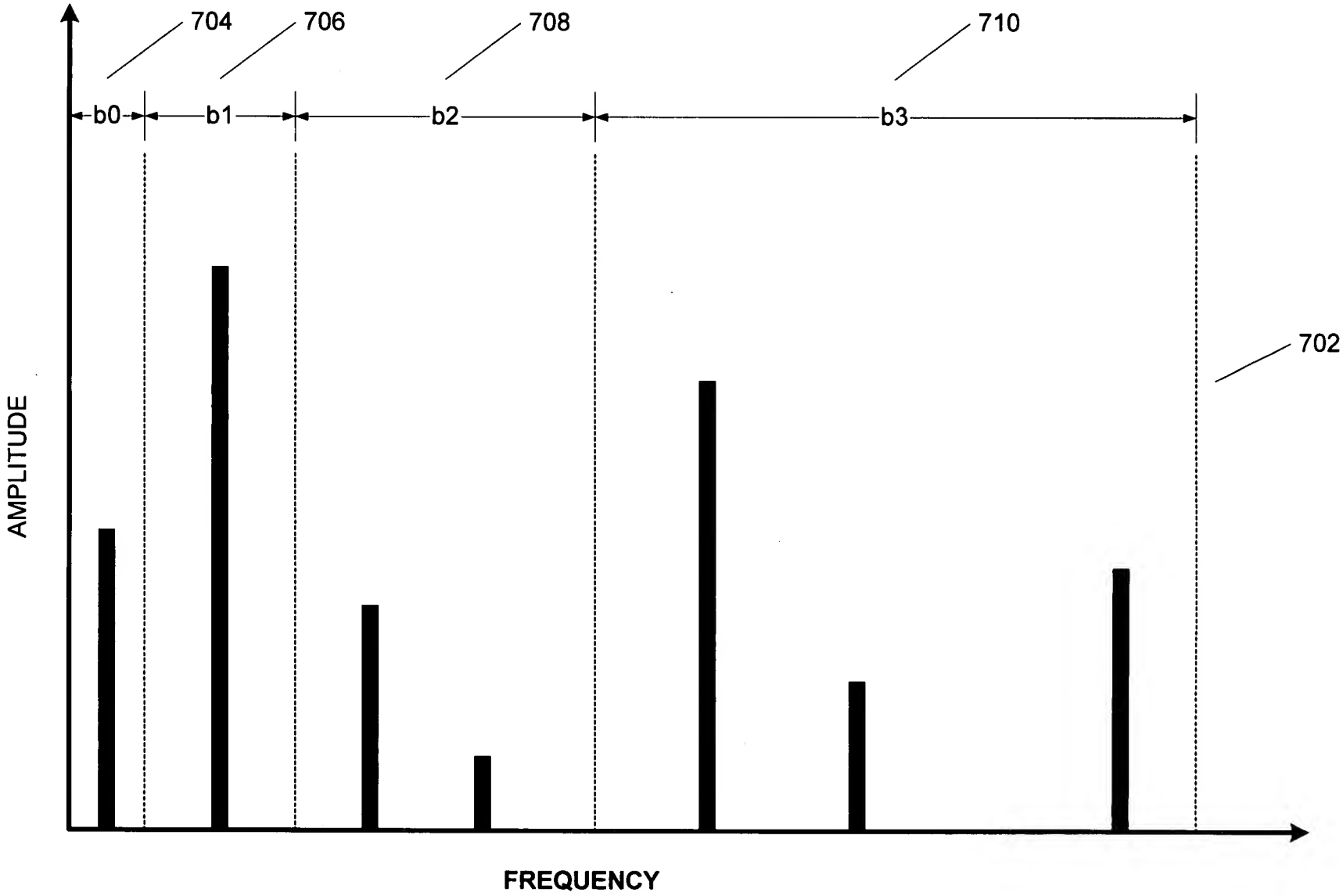

Overlapping amplitude modulated FFT time slice pre-filtering
t
604
606
+
0
-
AMPLITUDE
TIME
602

FIG. 7



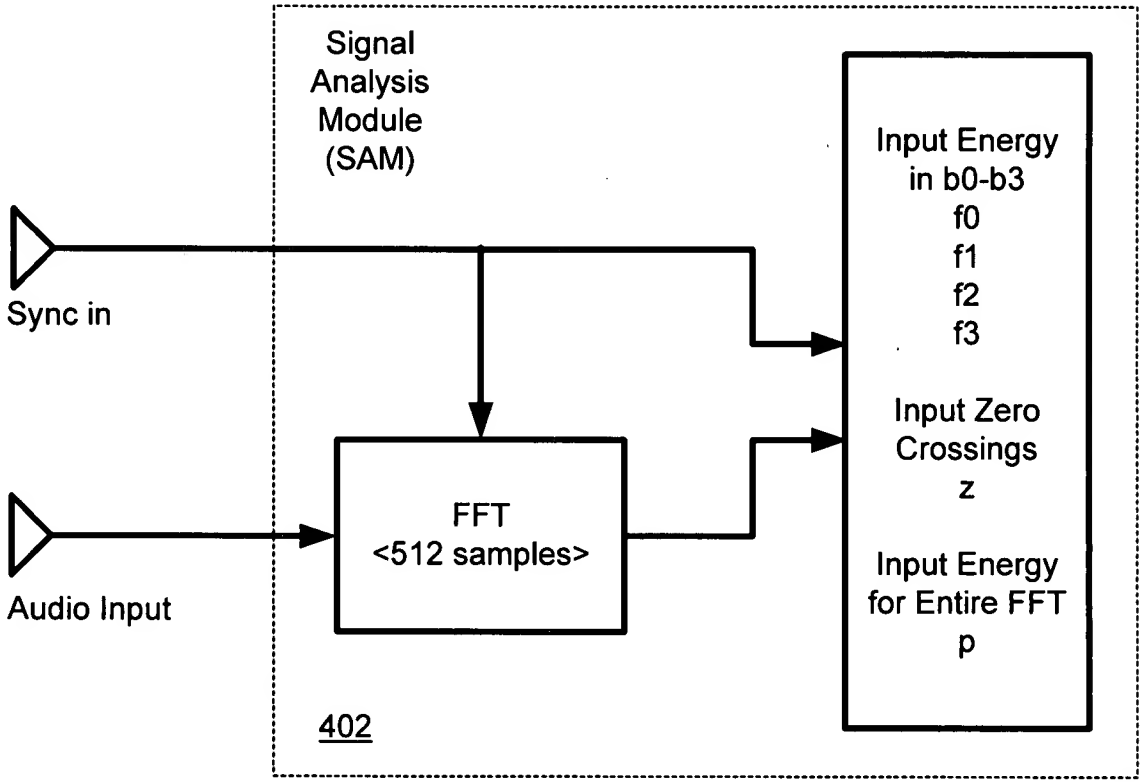

Signal
Analysis
Module
(SAM)
Sync in
Audio Input
FFT
<512 samples>
Input Energy
in b0-b3
f0
f1
f2
f3
Input Zero
Crossings
z
Input Energy
for Entire FFT
p
402

FIG. 8

FIG. 9A

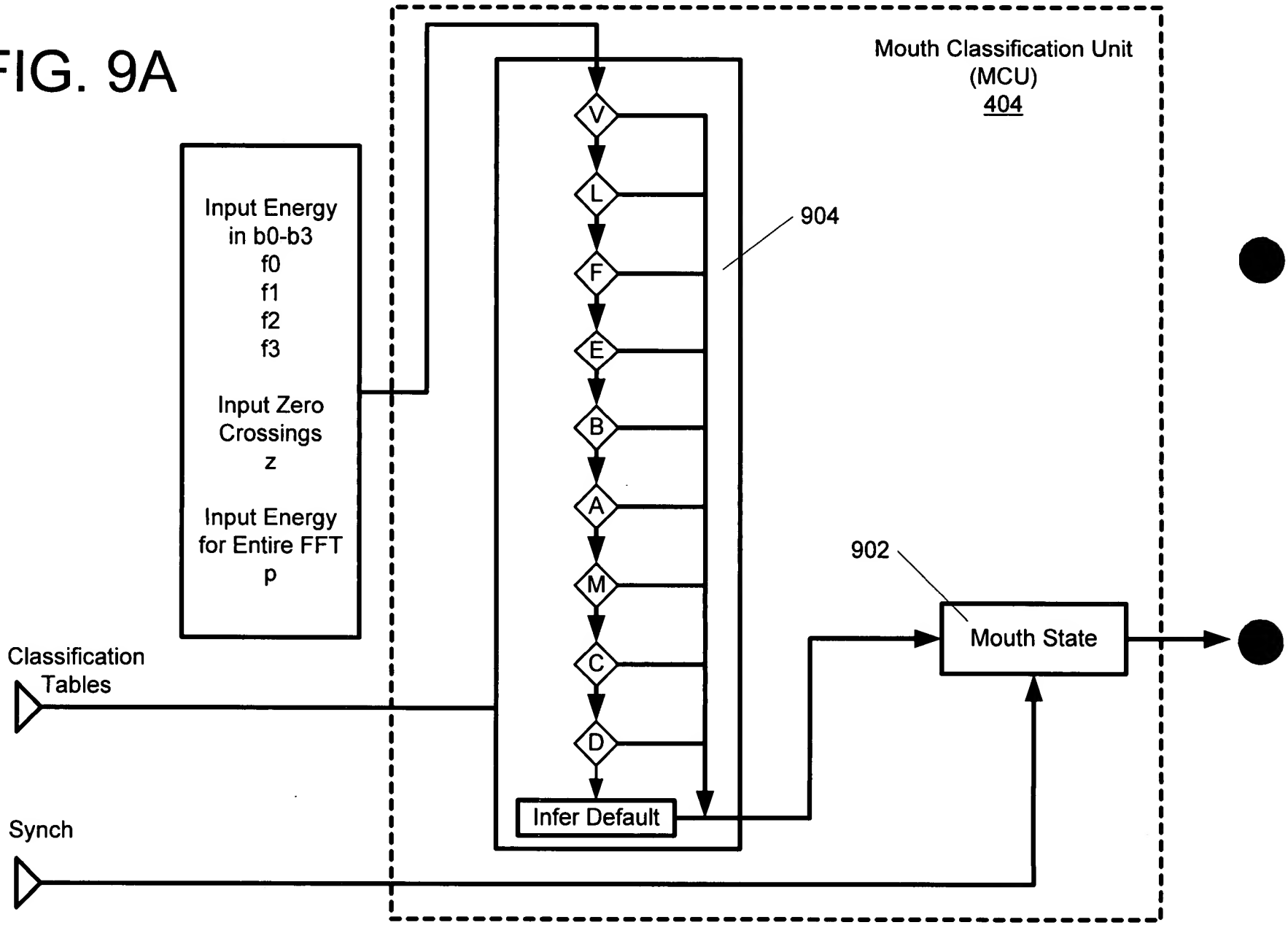
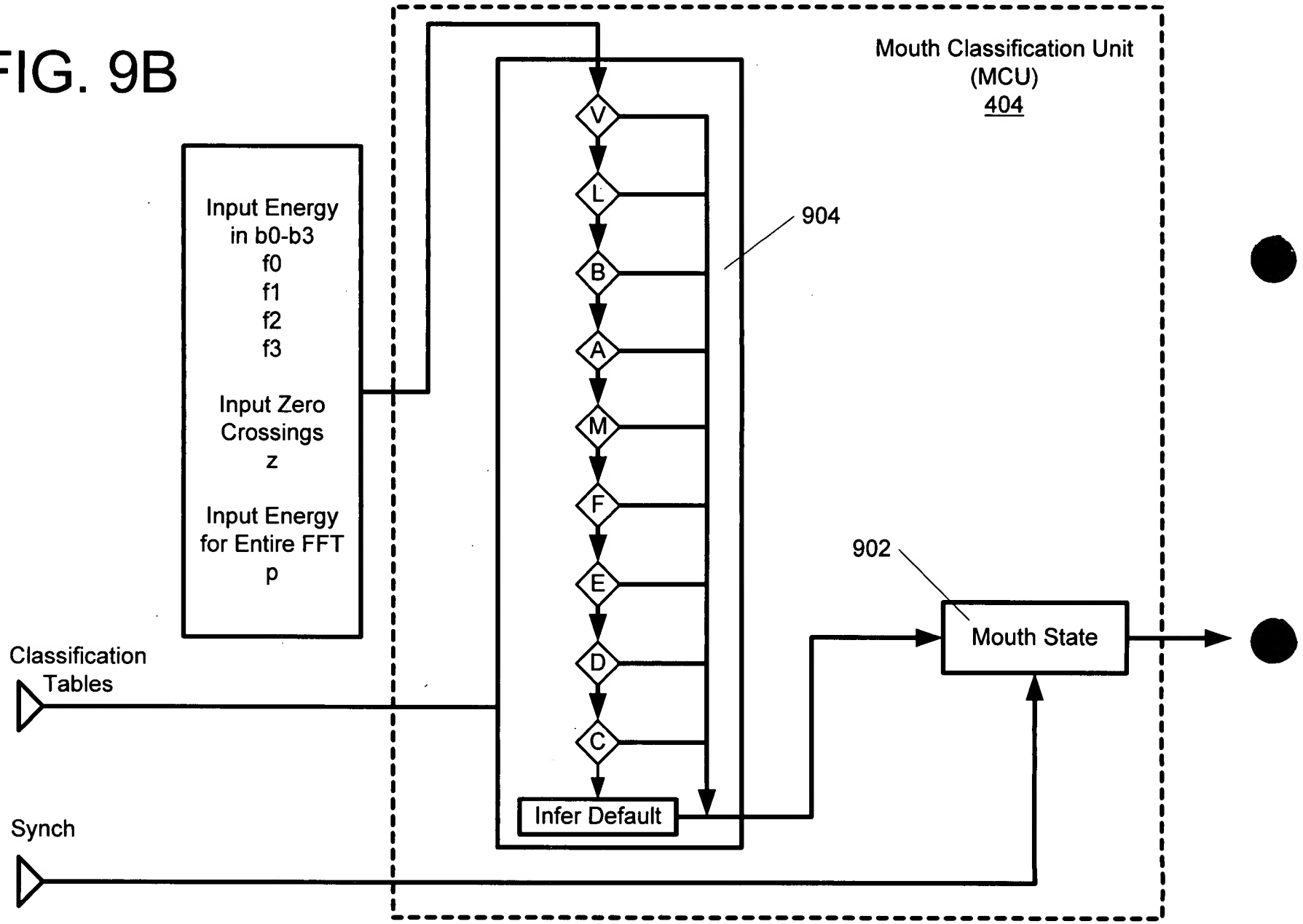

Mouth Classification Unit
(MCU)
404
Input Energy
in b0-b3
f0
f1
f2
f3
Input Zero
Crossings
z
Input Energy
for Entire FFT
p
V
L
F
E
B
A
M
C
D
Infer Default
904
902
Mouth State
Classification
Tables
Synch

FIG. 9B


Mouth Classification Unit
(MCU)
404
Input Energy
in b0-b3
f0
f1
f2
f3
Input Zero
Crossings
z
Input Energy
for Entire FFT
p
904
V
L
B
A
M
F
E
D
C
Infer Default
902
Mouth State
Classification
Tables
Synch

```
/* starting point for a Max external object ------- */

#include <A4Stuff.h>
#include <SetUpA4.h>

/* the required include files */
#include "ext.h"
#include <string.h>
#include <stdio.h>
#include <unistd.h>

#define Min(x,y) ((y <= x) ?  y : x)
#define Max(x,y)        ((y >= x) ?  y : x)
#define Abs(x)      ((x <  0) ? -x : x)

static long             instanceNum;
static long             logging = false;

typedef struct MCU
{
        Object  m_ob;                   // required
        FILE    *log;
        long    curr;
        long    prev;
        long    frame;
        long    power;
        long    zero;
        long    f0;
        long    f1;
        long    f2;
        long    f3;
        long    ppower;
        long    pzero;
```

FIG. 10A

```
        long    pf0;
        long    pf1;
        long    pf2;
        long    pf3;
        long    dbx;
        long    init;
        void    *out1;
} MCU;

/* global necessary for 68K function macros to work */
fptr *FNS;

/* globalthat holds the class definition */
void *MCU_class;

/* prototypes for your functions */

void *MCU_new  (void);
void *MCU_free (MCU *x);
void MCU_bang  (MCU *x);

void MCU_power          (MCU *x, long n);
void MCU_zero   (MCU *x, long n);
void MCU_f0     (MCU *x, long n);
void MCU_f1     (MCU *x, long n);
void MCU_f2     (MCU *x, long n);
void MCU_f3     (MCU *x, long n);
void MCU_dbx    (MCU *x, long n);

static int range    (long n, int bot, int top);

static int tryA (MCU *x);
static int tryB (MCU *x);
```

FIG. 10B

```
static int tryC (MCU *x);
static int tryD (MCU *x);
static int tryL (MCU *x);
static int tryE (MCU *x);
static int tryF (MCU *x);
static int tryV (MCU *x);
static int tryM (MCU *x);
```

FIG. 10C

```
/* initialization routine */

void main(fptr *f)
{

        EnterCodeResource();
        PrepareCallback();
        FNS = f;

        /* tell Max about your class. The cast to short is important for 68K */
        setup(&MCU_class, MCU_new, 0L, (short)sizeof(MCU), 0L, 0);

        /* bind your methods to symbols */
        addbang ((method) MCU_bang);

        addinx ((method) MCU_power, 1);
        addinx ((method) MCU_zero,  2);
        addinx ((method) MCU_f0,    3);
        addinx ((method) MCU_f1,    4);
        addinx ((method) MCU_f2,    5);
        addinx ((method) MCU_f3,    6);
        addinx ((method) MCU_dbx,       7);
```

```
        /* list object in the new object list */
        finder_addclass("Data","MCU");

        ExitCodeResource();

}

/* instance creation routine */

void    *MCU_new(void)
{
        MCU     *x;
        char    fname[256];

        x = (MCU *)newobject(MCU_class);

        intin (x, 7);
        intin (x, 6);
        intin (x, 5);
        intin (x, 4);
        intin (x, 3);
        intin (x, 2);
        intin (x, 1);

        x->out1 = intout (x);

        sprintf (fname, "MCULOG-%d", instanceNum++);

        x->log = fopen (fname, "w");
        if (x->log != 0)
                post (" -- MCU log file %s\n", fname);
        else
```

FIG. 10D

```
        post (" -- Can't open MCU log file %s\n", fname);

        x->frame        = 0;
        x->ppower       = 0;
        x->pzero        = 0;
        x->pf0          = 0;
        x->pf1          = 0;
        x->pf2          = 0;
        x->pf3          = 0;
        x->power        = 0;
        x->zero         = 0;
        x->f0           = 0;
        x->f1           = 0;
        x->f2           = 0;
        x->f3           = 0;
        x->dbx          = 0;
        x->init         = -1;


        return (x);
        }

void    *MCU_free (MCU *x)
{
        if (x->log != 0) {
                fclose (x->log);
                post ("MCU - EXIT");
                }
        return (x);
}

void MCU_bang (MCU *x)
{
```

FIG. 10E

```
        int mouth = -1;
        int found = -1;
        int was   = -1;
        int       now       = -1;

char    *failed = "******";
char    *ok     = ".....";

EnterCallback();


if (found == -1 ) {
        mouth = tryM (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryA (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryV (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryL (x);
        if (mouth != -1)
                found = mouth;
```

FIG. 10F

```
        }

if (found == -1 ) {
        mouth = tryB (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryF (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryE (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryD (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryC (x);
        if (mouth != -1)
                found = mouth;
        }

was = found;
```

FIG. 10G

```
        if (found == -1) {
                found = x->prev;
                }

        // transitions out of A/Z mouths need extra force
        if ((found != 0 && found != 1)  && (x->prev == 0 || x->prev == 1)) {
                if (x->power <= 20 && x->zero < 200)
                        found = 1;
                }

        if (x->frame >= 1) {
                outlet_int (x->out1, (long) found);
                if (x->dbx != 0) {
                        if (was == -1) {
                                post ("   %d [%d] %s %d -- %6d %4d -- %4d %4d %4d %4d" , found, x->frame, failed, was, x->power, x->zero, x-
>f0, x->f1, x->f2, x->f3);
                                fprintf (x->log, "[%5d] (%2d) (%2d) *** %6d %4d -- %4d %4d %4d %4d\n", x->frame, was, found, x->power, x-
>zero, x->f0, x->f1, x->f2, x->f3);
                                fflush (x->log);
                                }
                         if (was >= 0) {
                                post ("   %d [%d] %s %d -- %6d %4d -- %4d %4d %4d %4d" , found, x->frame, ok, was, x->power, x->zero, x->f0,
x->f1, x->f2, x->f3);
                                fprintf (x->log, "[%5d] (%2d) (%2d) *** %6d %4d -- %4d %4d %4d %4d\n", x->frame, was, found, x->power, x-
>zero, x->f0, x->f1, x->f2, x->f3);
                                fflush (x->log);
                                }
                        }
                 }


         x->frame++;
```

FIG. 10H

```
        x->prev = found;

        ExitCallback();

}

// the further down the try-chain the more latitude with power and zeros

int tryV (MCU *x)
{
        /*      VVV mouth       -       vee */
        if (range (x->power, 1, 20) && range (x->zero, 400, 700)) {
                if (range (x->f0, 10, 50) && range (x->f1, 10, 25) && range (x->f2, 10, 30) && range (x->f3, 10, 30)) {
                        return 8;
                        }
                }
        /*      VVV mouth       -       eff */
        if (range (x->power, 1, 6000) && range (x->zero, 150, 500)) {
                if (range (x->f0, 60, 93) && range (x->f1, 4, 50) && range (x->f2, 2, 40) && range (x->f3, 0, 90)) {
                        if (x->f2 < x->f1) {
                                return 8;
                                }
                        }
                }
        return (-1);
}


int tryL (MCU *x)
{
        /* LLL Mouth    -       tee     */
        if ((range (x->power, 5, 6000) && range (x->zero, 80, 450))) {
```

FIG. 10I

```
            if (range (x->f0, 0, 15) && range (x->f1, 0, 20) && range (x->f2, 30, 92) && range (x->f3, 10, 30)) {
                return 7;
                }
            }

        /* LLL Mouth    -       ndd     */
        if ((range (x->power, 5, 6000) && range (x->zero, 80, 250))) {
            if (range (x->f0, 0, 15) && range (x->f1, 0, 20) && range (x->f2, 30, 92) && range (x->f3, 0, 10)) {
                return 2;
                }
            }

        /* LLL Mouth    -       ell     */
        if ((range (x->power, 2, 2000) && range (x->zero, 200, 400))) {
            if (range (x->f0, 25, 85) && range (x->f1, 0, 16) && range (x->f2, 0, 10) && range (x->f3, 5, 75)) {
                return 7;
                }
            }

        return (-1);
}

int tryF (MCU *x)
{
        /* FFF Mouth    -       oo      */
        if (range (x->power, 2, 6000) && range (x->zero, 15, 120)) {
            if (range (x->f0, 97, 100) && range (x->f1, 0, 10) && range (x->f2, 0, 4) && range (x->f3, 0, 1)) {
                return 6;
                }
            }
        /* filters out transition ambigiuities */
        if (x->prev == 6) {
            if (range (x->f0, 93, 100) && range (x->f1, 0, 10) && range (x->f2, 0, 4) && range (x->f3, 0, 1)) {
```

FIG. 10J

FIG. 10K

```
                return 6;
                }
        }
    return (-1);
}

int tryE (MCU *x)
{
    /* EEE Mouth  -    er    */
    if (range (x->power, 1, 15000) && range (x->zero, 15, 110)) {
        if (range (x->f0, 40, 96) && range (x->f1, 0, 50) && range (x->f2, 0, 11) && range (x->f3, 0, 5)) {
            return 5;
            }
        }

    /* EEE Mouth  -    er    */
    if (range (x->power, 1, 15000) && range (x->zero, 30, 90)) {
        if (range (x->f0, 0, 60) && range (x->f1, 90, 100) && range (x->f2, 0, 11) && range (x->f3, 0, 5)) {
            return 5;
            }
        }

    return (-1);
}


int tryB (MCU *x)
{
    /*    BBB mouth    -    ee    */
    if (range (x->power, 1, 12000) && range (x->zero, 15, 400)) {
        if (range (x->f0, 10, 97) && range (x->f1, 0, 35) && range (x->f2, 1, 85) && range (x->f3, 0, 15)) {
            if ((x->f0 > (x->f1) * 2) && (x->f1 < x->f2) && (x->f2 > x->f3)) {
                return 2;
```

```
                        }
                    }
                }
        /*      BBB mouth       -       ess     */
        if (range (x->power, 1, 12000) && range (x->zero, 200, 650)) {
                if (range (x->f0, 0, 50) && range (x->f1, 0, 5) && range (x->f2, 0, 80) && range (x->f3, 0, 100)) {
                        if ((x->f3 > x->f1) && (x->f2 > x->f1)) {
                                return 2;
                                }
                        }
                }
        return (-1);
}

int tryA (MCU *x)       // M-B-P mouths (zero power but could have mid-high zero (breaths) no spectral content)
{
        /*      AAA mouth       -       bee, pee        */
        if (range (x->power, 1, 2) && range (x->zero, 0, 600)) {
                if (range (x->f0, 0, 0) && range (x->f1, 0, 0) && range (x->f2, 0, 0) && range (x->f3, 0, 0)) {
                        return 1;
                        }
                }

        /*      AAA mouth       -       bee, pee        */
        if (range (x->power, 1, 2) && range (x->zero, 0, 600)) {
                if (range (x->f0, 0, 40) && range (x->f1, 0, 10) && range (x->f2, 0, 10) && range (x->f3, 0, 0)) {
                        return 1;
                        }
                }

        /*      AAA mouth       -       emm     */
        if (range (x->power, 1, 20) && range (x->zero, 10, 100)) {
```

FIG. 10L

```
                if (range (x->f0, 50, 95) && range (x->f1, 0, 40) && range (x->f2, 0, 10) && range (x->f3, 0, 5)) {
                        return 1;
                        }
                }
        return (-1);
}

int tryM (MCU *x)       // M mouth (breaths, tape noise, quiet - non-MBP)
{
        /*      MMM mouth       -       tape noise      return ZZZ mouth*/
        if (range (x->power, 0, 1) && range (x->zero, 0, 1000)) {
                if (range (x->f0, 0, 10) && range (x->f1, 0, 1) && range (x->f2, 0, 1) && range (x->f3, 0, 1)) {
                        return 0;
                        }
                }

        /*      MMM mouth       -       breaths */
        if (range (x->power, 1, 5) && range (x->zero, 1, 60)) {
                if (range (x->f0, 0, 50) && range (x->f1, 0, 10) && range (x->f2, 0, 5) && range (x->f3, 0, 1)) {
                        return 9;
                        }
                }


        return (-1);
}

int tryC (MCU *x)
{
        /* CCC Mouth    -       EH */
        if (range (x->power, 1, 15000) && range (x->zero, 30, 200)) {
                if (range (x->f0, 10, 80) && range (x->f1, 10, 80) && range (x->f2, 0, 20) && range (x->f3, 0, 5)) {
                        return 3;
```

FIG. 10M

```
                }
        }

    /*      CCC mouth       -       AH */
    if (range (x->power, 1, 15000) && range (x->zero, 30, 225)) {
        if (range (x->f0, 0, 50) && range (x->f1, 10, 99) && range (x->f2, 0, 60) && range (x->f3, 0, 10)) {
            return 3;
            }
        }

    return (-1);
}

int tryD (MCU *x)
{
    /*      DDD mouth       -       EH */
    if (range (x->power, 15000, 40000) && range (x->zero, 30, 150)) {
        if (range (x->f0, 10, 80) && range (x->f1, 10, 80) && range (x->f2, 0, 20) && range (x->f3, 0, 5)) {
            return 4;
            }
        }

    /*      DDD mouth       -       AH */
    if (range (x->power, 15000, 40000) && range (x->zero, 30, 180)) {
        if (range (x->f0, 0, 30) && range (x->f1, 10, 99) && range (x->f2, 0, 60) && range (x->f3, 0, 10)) {
            return 4;
            }
        }
    return (-1);
}

void MCU_power      (MCU *x, long n)
```

FIG. 10N

```
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->power = x->power;
                x->ppower = n;
                }
        else {
                x->power = (x->ppower + n) / 2;
                x->ppower = n;
                }


        ExitCallback();
}

void MCU_zero   (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->zero = x->zero;
                x->pzero = n;
                }
        else {
                x->zero = (x->pzero + n) / 2;
                x->pzero = n;
                }


        ExitCallback();
}
```

FIG. 10O

```
void MCU_f0    (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f0 = x->f0;
                x->pf0 = n;
                }
        else {
                x->f0 = (x->pf0 + n) / 2;
                x->pf0 = n;
                }


        ExitCallback();
}

void MCU_f1    (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f1 = x->f1;
                x->pf1 = n;
                }
        else {
                x->f1 = (x->pf1 + n) / 2;
                x->pf1 = n;
                }


        ExitCallback();
}
```

FIG. 10P

```
void MCU_f2    (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f2 = x->f2;
                x->pf2 = n;
                }
        else {
                x->f2 = (x->pf2 + n) / 2;
                x->pf2 = n;
                }

        ExitCallback();
}

void MCU_f3    (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f3 = x->f3;
                x->pf3 = n;
                }
        else {
                x->f3 = (x->pf3 + n) / 2;
                x->pf3 = n;
                }

        ExitCallback();
}
```

FIG. 10Q

```
void MCU_dbx    (MCU *x, long n)
{
        EnterCallback();

        x->dbx = n;

        if (n == 1 && x->log != 0) {
                post ("Logging: ON");
                x->frame = 0;
                }
        else
                post ("Logging: OFF");


        ExitCallback();
}


static int range (long n, int bot, int top)
{
        if (n >= bot && n <= top)
                return 1;
        else
                return 0;
}
```

FIG. 10R

FIG. 10S

```
/* starting point for a Max external object ------- */

#include <A4Stuff.h>
#include <SetUpA4.h>

/* the required include files */
#include "ext.h"
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>

#define Min(x,y)  ((y <= x) ?  y : x)
#define Max(x,y)  ((y >= x) ?  y : x)
#define Abs(x)       ((x <  0) ? -x : x)

static long                instanceNum;
static long                logging = false;

typedef enum erange {ALL, Z, VL, L, ML, M, MH, H, VH, X} Erange;

typedef struct MCU
{
        Object   m_ob;                          // required
        FILE     *log;
        long     curr;
        long     prev;
        long     frame;
        long     ppow;                          // used in finalCheck()
        long     power;
        long     zero;
        long     f0;
        long     f1;
        long     f2;
        long     f3;
        long     ppower;
        long     pzero;
```

```
        long    pf0;
        long    pf1;
        long    pf2;
        long    pf3;
        long    dbx;
        long    init;
        void    *out1;
} MCU;

/* global necessary for 68K function macros to work */
fptr *FNS;

/* globalthat holds the class definition */
void *MCU_class;

/* prototypes for your functions */

void *MCU_new  (void);
void *MCU_free (MCU *x);
void MCU_bang  (MCU *x);

void MCU_power  (MCU *x, long n);
void MCU_zero   (MCU *x, long n);
void MCU_f0     (MCU *x, long n);
void MCU_f1     (MCU *x, long n);
void MCU_f2     (MCU *x, long n);
void MCU_f3     (MCU *x, long n);
void MCU_dbx    (MCU *x, long n);

static int    range  (long n, int bot, int top);    // low level range classification

static Erange franger (long f);                     // high level range classification return enun
static Erange zranger (long z);                     // high level range classification return enum
static Erange pranger (long p);                     // high level range classification return enum

static int   matcher (long p, long z, long f0, long f1, long f2, long f3);
```

FIG. 10T

```
static int   matchf (long f0, long f1, long f2, long f3);
static int   matchz (long lz, long hz);
static int   matchp (long lp, long hp);

static int   finalCheck (int candidate);

static MCU*        gState;

static int tryA (MCU *x);
static int tryB (MCU *x);
static int tryC (MCU *x);
static int tryD (MCU *x);
static int tryL (MCU *x);
static int tryE (MCU *x);
static int tryF (MCU *x);
static int tryV (MCU *x);
static int tryM (MCU *x);

/* initialization routine */

void main(fptr *f)
{

        EnterCodeResource();
        PrepareCallback();
        FNS = f;

        post ("MCU-bunny fall v1.0");

        /* tell Max about your class. The cast to short is important for 68K */
        setup(&MCU_class, MCU_new, 0L, (short)sizeof(MCU), 0L, 0);

        /* bind your methods to symbols */
        addbang ((method) MCU_bang);

        addinx ((method) MCU_power, 1);
        addinx ((method) MCU_zero,  2);
```

FIG. 10U

```
        addinx ((method) MCU_f0,    3);
        addinx ((method) MCU_f1,    4);
        addinx ((method) MCU_f2,    5);
        addinx ((method) MCU_f3,    6);
        addinx ((method) MCU_dbx,           7);


        /* list object in the new object list */
        finder_addclass("Data","MCU");


        ExitCodeResource();

}


/* instance creation routine */

void    *MCU_new(void)
{
        MCU     *x;
        char    fname[256];

        x = (MCU *)newobject(MCU_class);

        intin (x, 7);
        intin (x, 6);
        intin (x, 5);
        intin (x, 4);
        intin (x, 3);
        intin (x, 2);
        intin (x, 1);

        x->out1 = intout (x);

        sprintf (fname, "MCULOG-%d", instanceNum++);
```

FIG. 10V

```
        x->log = fopen (fname, "w");
        if (x->log != 0)
                post (" -- MCU log file %s\n", fname);
        else
                post (" -- Can't open MCU log file %s\n", fname);

        x->prev    = 0;
        x->curr    = 0;
        x->ppow    = 0;

        x->frame        = 0;
        x->ppower       = 0;
        x->pzero= 0;
        x->pf0          = 0;
        x->pf1          = 0;
        x->pf2          = 0;
        x->pf3          = 0;
        x->power        = 0;
        x->zero         = 0;
        x->f0           = 0;
        x->f1           = 0;
        x->f2           = 0;
        x->f3           = 0;
        x->dbx          = 0;
        x->init         = -1;


        return (x);
        }

void    *MCU_free (MCU *x)
{
        if (x->log != 0) {
                fclose (x->log);
                post ("MCU - EXIT");
                }
        return (x);
```

FIG. 10W

```
}

void MCU_bang (MCU *x)
{
        int mouth = -1;
        int found = -1;
        int was   = -1;
        int       now       = -1;
        int If    =  0;

        char      *failed = "******";
        char      *ok     = ". . . . .";

        EnterCallback();


        if (found == -1 ) {
                mouth = tryV (x);
//              if (mouth != -1)
//                      found = mouth;
                }

        if (found == -1 ) {
                mouth = tryL (x);
//              if (mouth != -1)
//                      found = mouth;
                }

        if (found == -1 ) {
                mouth = tryB (x);
                if (mouth != -1)
                        found = mouth;
                }

        if (found == -1 ) {
                mouth = tryA (x);
                if (mouth != -1)
```

FIG. 10X

```
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryM (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryF (x);
//      if (mouth != -1)
//              found = mouth;
        }

if (found == -1 ) {
        mouth = tryE (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryD (x);
        if (mouth != -1)
                found = mouth;
        }

if (found == -1 ) {
        mouth = tryC (x);
        if (mouth != -1)
                found = mouth;
        }

was = found;

if (found == -1) {
        found = 3;
```

FIG. 10Y

```
		}


	// transitions out of A/Z mouths need extra force
	//if ((found != 0 && found != 1)  && (x->prev == 0 || x->prev == 1)) {
	//		if (x->power <= 20 && x->zero < 200)
	//			found = 1;
	//		}

	if (x->frame >= 1) {
			outlet_int (x->out1, (long) found);
			if (x->dbx != 0) {
				if (was == -1) {
					post ("   %d [%d] %s %d -- %6d %4d -- %4d %4d %4d %4d" , found, x->frame, failed, was, x->power, x->zero, x->f0, x->f1, x-
>f2, x->f3);
					fprintf (x->log, "[%5d] (%2d) (%2d) *** %6d %4d -- %4d %4d %4d %4d\n", x->frame, was, found, x->power, x->zero, x->f0, x-
>f1, x->f2, x->f3);
					fflush (x->log);
					}
				if (was >= 0) {
					post ("   %d [%d] %s %d -- %6d %4d -- %4d %4d %4d %4d" , found, x->frame, ok, was, x->power, x->zero, x->f0, x->f1, x->f2,
x->f3);
					fprintf (x->log, "[%5d] (%2d) (%2d) *** %6d %4d -- %4d %4d %4d %4d\n", x->frame, was, found, x->power, x->zero, x->f0, x-
>f1, x->f2, x->f3);
					fflush (x->log);
					}
				}
			}


	x->frame++;
	x->prev = found;
	x->ppow = x->power;

	ExitCallback();

}
```

FIG. 10Z

```
int tryV (MCU *x)
{
        gState = x;

        /*      VVV mouth       -       eff             */

        if (matchf (M, M, L, Z) || matchf (M, M, Z, Z))
                if (matchz (VH, VH) && matchp (VL, L))
                        return 8;

        /*      VVV mouth       -       vvv             */

        if (matchf (X, Z, Z, Z) || matchf (VH, Z, Z, Z))
                if (matchz (VL, VL) && matchp (VL, L))
                        return 8;

        return (finalCheck (8));
}

int tryL (MCU *x)
{
        gState = x;

        /* LLL Mouth    -       elll            */

        if (matchf (H, M, Z, Z) || matchf (H, L, Z, Z))
                if (matchz (M, M) && matchp (L, H))
                        return 7;

        /* LLL Mouth    -       alll            */

        if (matchf (M, M, Z, Z) || matchf (M, L, Z, Z))
                if (matchz (M, M) && matchp (L, H))
                        return 7;
```

FIG. 10AA

```
        if (matchf (M, L, L, L) || matchf (M, L, Z, L))
                if (matchz (M, M) && matchp (L, H))
                        return 7;

        if (matchf (M, Z, L, Z) || matchf (M, Z, Z, Z))
                if (matchz (M, M) && matchp (L, H))
                        return 7;

        return (finalCheck (7));
}

int tryF (MCU *x)
{
        if (matchz (VL, VL))
                return 6;

        return (finalCheck (6));
}

int tryE (MCU *x)
{
        if (matchf (L, L, Z, Z)) {
                if (matchz (L, M) && matchp (L, H))
                        return 5;
                        }

        return (finalCheck (5));
}


int tryB (MCU *x)
{
        gState = x;

        /*      BBB mouth       -       tsss    */
```

FIG. 10BB

```
        if (matchz (H, H) && matchp (VL, M))
                return 2;

        if (matchf (Z, Z, Z, L) || matchf (Z, Z, Z, M) || matchf (Z, Z, Z, H)) {
                if (matchz (H, H) && matchp (VL, M))
                        return 2;
                        }
        if (matchf (Z, Z, VL, L) || matchf (Z, Z, VL, M) || matchf (Z, Z, VL, H)) {
                if (matchz (H, H) && matchp (VL, M))
                        return 2;
                        }
        if (matchf (Z, VL, VL, L) || matchf (Z, VL, VL, M) || matchf (Z, VL, VL, H)) {
                if (matchz (H, H) && matchp (VL, M))
                        return 2;
                        }
        return (finalCheck (2));
}

int tryA (MCU *x) // M-B-P mouths
{
        gState = x;


        if (matchf (M, Z, Z, Z)) {                                      /* MMM */
                if (matchz (VL, VL) && matchp (VL, X))
                        return 1;
                        }

        return (-1);                    // dont finalCheck()
}

int tryM (MCU *x) // M mouth (breaths, tape noise, quiet - non-MBP)
{
        gState = x;

        if (matchf (Z, Z, Z, Z) || matchz (Z, Z))                       /* silence */
```

FIG. 10CC

```
                return 9;

        if (matchp (Z, VL) && matchz (VL, X))                    /* breath */
                return 9;


        return (-1);                    // dont finalCheck()
}

int tryC (MCU *x)
{
        gState = x;

        if (matchp (L, M))
                return 3;

        return (finalCheck (3));
}

int tryD (MCU *x)
{
        gState = x;

        if (matchp (M, X))
                return 4;

        return (finalCheck (4));
}


void MCU_power  (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->power = x->power;
                x->ppower = n;
```

FIG. 10DD

```
                }
        else {
                x->power = (x->ppower + n) / 2;
                x->ppower = n;
                }


        ExitCallback();
}

void MCU_zero   (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->zero = x->zero;
                x->pzero = n;
                }
        else {
                x->zero = (x->pzero + n) / 2;
                x->pzero = n;
                }


        ExitCallback();
}

void MCU_f0     (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f0 = x->f0;
                x->pf0 = n;
                }
        else {
                x->f0 = (x->pf0 + n) / 2;
```

FIG. 10EE

```
                x->pf0 = n;
                }


        ExitCallback();
}

void MCU_f1     (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f1 = x->f1;
                x->pf1 = n;
                }
        else {
                x->f1 = (x->pf1 + n) / 2;
                x->pf1 = n;
                }


        ExitCallback();
}

void MCU_f2     (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f2 = x->f2;
                x->pf2 = n;
                }
        else {
                x->f2 = (x->pf2 + n) / 2;
                x->pf2 = n;
                }
```

FIG. 10FF

```
        ExitCallback();
}

void MCU_f3    (MCU *x, long n)
{
        EnterCallback();

        if ((x->frame % 2) == 0) {
                x->f3 = x->f3;
                x->pf3 = n;
                }
        else {
                x->f3 = (x->pf3 + n) / 2;
                x->pf3 = n;
                }

        ExitCallback();
}

void MCU_dbx    (MCU *x, long n)
{
        EnterCallback();

        x->dbx = n;

        if (n == 1 && x->log != 0) {
                post ("Logging: ON");
                x->frame = 0;
                }
        else
                post ("Logging: OFF");


        ExitCallback();
}

static int range (long n, int bot, int top)
```

FIG. 10GG

```
{
        if (n >= bot && n <= top)
                return 1;
        else
                return 0;
}

static int finalCheck (int candidate)
{
        long change, small, power, ppow;

        int match = -1;

        if (candidate == gState->prev) {
                power  = gState->power;
                ppow   = gState->ppow;

                if (power < 25)
                        small = 10;
                else if (power <  50)
                        small = 15;
                else if (power <  100)
                        small = 20;
                else if (power <  300)
                        small = 50;
                else if (power < 1000)
                        small = 100;
                else if (power < 3000)
                        small = 150;
                else
                        small = 200;


                change = Abs ((power - ppow));


                if (change < small){
```

FIG. 10HH

```
                        match = candidate;
                        // post ("final Check (%d): %d, %d - %d %d", candidate, small, change, ppow, power);
                        }

                }
        return match;
}

// classify formant ranges as High, Med, Low &c.

Erange franger (long f)
{
        if (range (f, 0, 5))
                return Z;

        if (range (f, 5, 11))
                return VL;

        if (range (f, 11, 40))
                return L;

        if (range (f, 40, 70))
                return M;

        if (range (f, 70, 92))
                return H;

        if (range (f, 93, 97))
                return VH;

        if (range (f, 98, 100))                 // N.B. X is used here when MAX extends downward
                return X;

}

// classify zero crossing ranges as High, Med, Low &c.
```

FIG. 10II

```
Erange zranger (long z)
{
        if (range (z, 0, 5))
                return Z;

        if (range (z, 5, 20))
                return VL;

        if (range (z, 21, 43))
                return L;

        if (range (z, 43, 120))
                return M;

        if (range (z, 120, 250))
                return H;

        if (range (z, 250, 500))
                return VH;

        if (range (z, 500, 2000))
                return X;

}

// classify power ranges as High, Med, Low &c.

Erange pranger (long p)
{
        if (range (p, 0, 2))
                return Z;

        if (range (p, 3, 35))
                return VL;

        if (range (p, 35, 300))
                return L;
```

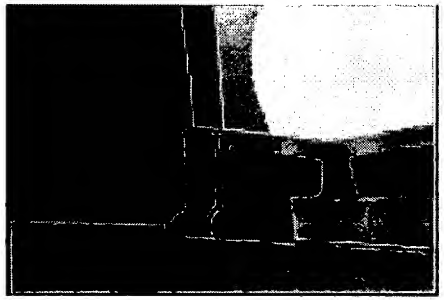
FIG. 10JJ

```
        if (range (p, 300, 1000))
                return ML;

        if (range (p, 1000, 4000))
                return M;

        if (range (p, 4000, 8000))
                return H;

        if (range (p, 8000, 10000))
                return VH;

        if (range (p, 10000, 100000))
                return X;

}


int matchf (long f0, long f1, long f2, long f3)
{
        MCU *x = gState;

        if (franger (x->f0) != f0)
                return 0;

        if (franger (x->f1) != f1)
                return 0;

        if (franger (x->f2) != f2)
                return 0;

        if (franger (x->f3) != f3)
                return 0;

        return 1;
}
```

FIG. 10KK

```
int matchz (long lz, long hz)
{
        int     z = (int) zranger (gState->zero);

        if (z < lz || z > hz)
                return 0;

        return 1;
}

int matchp (long lp, long hp)
{
        int p = (int) pranger (gState->power);

        if (p < lp || p > hp)
                return 0;

        return 1;
}
```

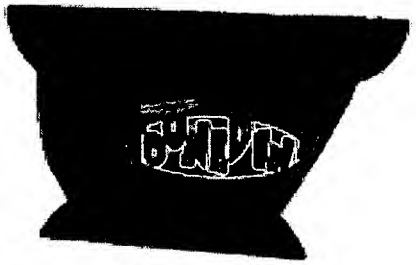
FIG. 10LL



Scene Description

background

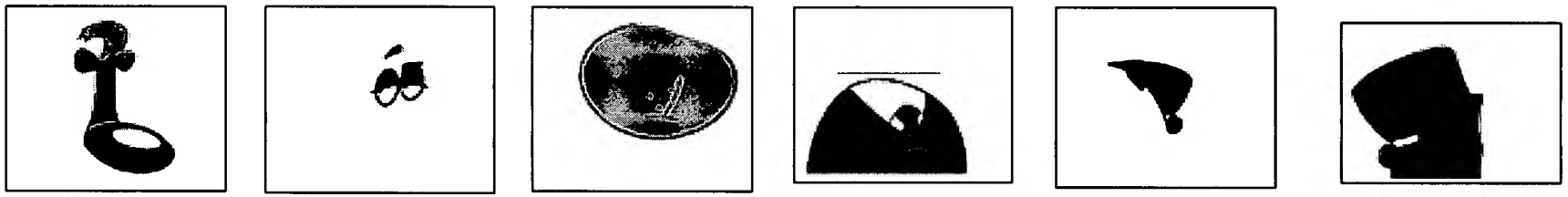
foreground



1102

1118

1106 1108 1110 1112 1114 1106



head eyes mouth body left arm right arm

Character

1104

FIG. 11

Pose A
Character: Buddy

Pose C
Character: Buddy

FIG. 12

Character Table

Pose B
Character: Buddy

Pose D
Character: Buddy

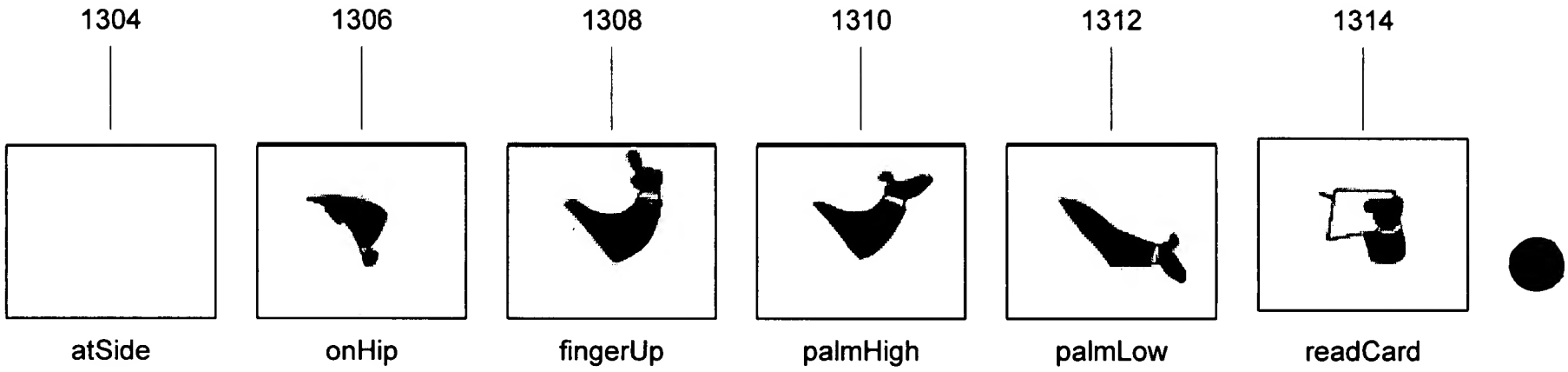


FIG. 13

TwoShot	Buddy	Pose A	LeftArm	-	Gestures
scene	*character*	*pose*	*element*		

TwoShot : Buddy : Pose A	(*partial*)					
LeftArm:	atSide	onHip	fingerUp	palmHigh	palmLow	readCard
RightArm:	atSide	onHip		palmHigh		
Head:	profile-L	3/4Left	front	3/4Front	profile-R	
Mouth:	down	up				
Eyes:	wide	halfmast				

1302

Sequence Table (in Pose transition)

1402

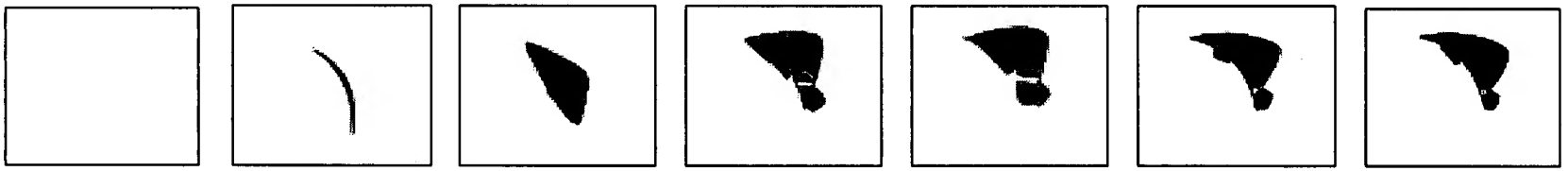


Image 1	Image 2	Image 3	Image 4	Image 5	Image 6	Image 7
TwoShot *Scene*	Buddy *character*	Pose A *pose*	LeftArm *element*	atSide *fromGesture*	to	onHip *toGesture*

FIG. 14A

Sequence Table (Pose to Pose transition)

1404

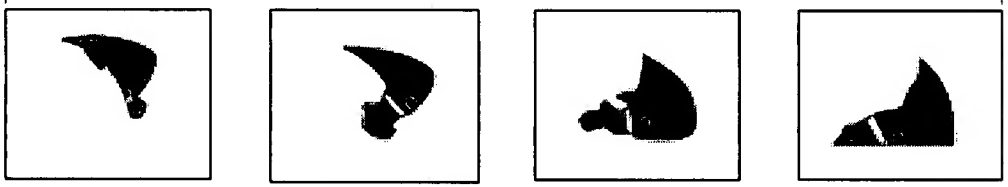


Image 7	Image 101	Image 102	Image 103
TwoShot *Scene*	Buddy *character*	Pose A *pose*	LeftArm *element*	onHip *fromGesture*
		TO		
TwoShot *Scene*	Buddy *character*	Pose B *pose*	LeftArm *element*	onDesk *toGesture*

FIG. 14B

Transition Table
(partial)

Table entries in cells may include: other commands, program segments, control information, indirect references, geometric transformations (i.e. rotation, translation and scale), tags, magnification or zoom factors, functions, parameters (i.e. lighting, shading, color, color change, layer shift, gradient, warp, bend, twist), or combinations thereof

<character> Buddy

<state> Pose-A : LeftArm - atSide

	Pose-A : LeftArm - atSide							
In-Pose transitions	Pose-A : LeftArm - onHip	1	2	3	4	5	6	7
	Pose-A : LeftArm - fingerUp							
	Pose-A : LeftArm - palmHigh							
	Pose-A : LeftArm - palmLow							
	Pose-A : LeftArm - readCard							
	class-1							
	default							
Pose-Pose transitions	Pose-B: LeftArm - atSide							
	Pose-B : LeftArm - onDesk	7	101	102	103			
	Pose-B : LeftArm - tapDesk							
	Pose-B : LeftArm - pointWest							
	class-1							
	default							

1502

FIG. 15

a) 4

b) <frame ID>

c) *x,y*

d) $\Delta x, \Delta y, \Delta z$

e) r, θ

f) r, θ, ϕ

g) <tag>

h) <parameter>

i) *zoom* μ

j) $\Delta x, \Delta y$, zoom μ, <parameter>, <tag>

k) <program segment start pointer>

l)

```
LOAD CONTROL_3, 5, X
        WAITFOR event=timer
        getObject (f)
                .
                .
                .

CONTINUE
```

FIG. 16

```
/* pics.c -- PICS file animation in a window using Sprites */
/*
        Mile 1 was creation of mpics objects based on pics object
        Mile 2 compositing with blue background
        Mile 3 compositing with masks

        Bug fixes 11/5 - 11/10 file not found use -M, sterns eyes,matte bug
        11/11 Move BG          ================>look into efficiancy
        11/18,19/98 fixed crashing of going outside of bg with clipping in PM5Copy.
        ---------->CAN BE OPTOMIZED MORE,some done 12/8/98mrn
        through 12 - 1 - 98 Continued work on large file crashes.  FLIP FLOP and FLIP and FLOP impl
        through 12-??? work on dirty rects, large file loading
*/

#include "QDOffScreen.h"
#include "ext.h"
#include "ext_common.h"
#include "ext_wind.h"
#include <Palettes.h>
#include "ext_anim_mrn.h"
#include "ext_strings.h"
#include <SetUpA4.h>
#include <A4Stuff.h>
#include <Movies.h>
#include <Math.h>
#include <Retrace.h>

// MPICS .h
#include "mpicsdef.h"
#include "mpicsproto.h"


fptr *FNS;
void *pics_class;
//short numPaletteColors;
short hasColorQD;
Symbol *ps_pics;
int inMask;

void main(fptr *f)
{
        CInfoRec ct;

        EnterCodeResource();
```

FIG. 17

```
	PrepareCallback();
	FNS = f;

	setup(&pics_class, pics_new, (method)pics_free, (short)sizeof(Pics), 0L, A_GIMME, 0);
	addbang((method)pics_bang);
	addint((method)pics_int);
//	addinx((method)pics_in1,1);
//	addinx((method)pics_in2,2);
	addmess ((method)pics_update, "update", A_CANT, 0);
	addmess ((method)pics_activate, "activate", A_CANT, 0);
	addmess ((method)pics_close, "close", A_CANT, 0);

	addmess((method)pics_priority, "priority", A_GIMME, 0);
	addmess((method)pics_assist,"assist",A_CANT,0);
	addmess((method)pics_test,"test",A_LONG,0);
	addmess((method)TestMethod,"mm",A_GIMME,0);
	addmess((method)pics_Loadit,"loadit",A_GIMME,0);
	finder_addclass("Graphics",MPICSX);
	addmess((method)pics_setxy,"setxy",A_GIMME,0);
	addmess((method)pics_winxy,"winxy",A_GIMME,0);
	addmess((method)pics_movxy,"movxy",A_GIMME,0);
	addmess((method)pics_BGxy,"BGxy",A_GIMME,0);

	addmess((method)pics_bbang,"bigbang",A_LONG,0);

	addmess((method)pics_hide,"hide",A_GIMME,0);
	addmess((method)pics_show,"show",A_GIMME,0);
	addmess((method)pics_next,"next",A_GIMME,0);
	addmess((method)pics_prev,"prev",A_GIMME,0);
	addmess((method)pics_whatframe,"whatframe",A_GIMME,0);
	addmess((method)pics_globxy,"globxy",A_GIMME,0);
	addmess((method)pics_setframe,"setframe",A_GIMME,0);
	addmess((method)pics_getmovie,"getmovie",A_GIMME,0);
	addmess((method)pics_closeall,"closeall",A_GIMME,0);
	addmess((method)pics_startmovie,"startmovie",A_GIMME,0);
	addmess((method)pics_stopmovie,"stopmovie",A_GIMME,0);
	addmess((method)pics_setmovieframe,"setmovieframe",A_GIMME,0);
	addmess((method)pics_front,"front",A_GIMME,0);

	alias("pics2");
	//finder_addclass("Graphics","pics2");
	ps_pics = gensym(MPICSX);
	colorinfo(&ct);
	hasColorQD = ct.c_hasColorQD;
```

FIG. 18

```
	//numPaletteColors = ct.c_depth==8? 256 : (ct.c_depth==4? 16 : 2);
	rescopy('STR#',3011);
	post("MPICS 4.9v");
	ExitCodeResource();
}

int Alias2Index(Pics *x, char *c)
{
	int i;
	for(i=0;i<=x->count;i++)
		{
		if(strcmp(x->m[i].AliasName,c)==0)
			return(i);
		}
	post("mpics: %s not valid alias",c);
	return -1;
}

int MGAlias2Index(Pics *x, char *c)
{
	int i;

	for(i=0;i<=x->MGCount;i++)
		{
		if(strcmp(x->MG[i].MMAlias,c)==0)
			return(i);
		}
	post("mpics: %s not valid MOVIE alias",c);
	return -1;
}

void pics_bang(Pics *x)
{
	int i;
	int frame;
	Rect sr;
	EnterCallback();

	if(x->lock)
		{
		error("LOCKED QFN");
		goto hell;
		}
```

FIG. 19

```
	for(i=0;i<=x->count;i++)
		{
		if(x->m[i].dirty == FIRST_X)//
			{
			x->m[i].dirty = 1;
			}
		}

	for(i=0;i<=x->count;i++)
		{
		if(x->m[i].visible == 0)
			continue;

		if(x->m[i].dirty == 1)//
			{
			//post("dirty %s",x->m[i].AliasName);

			frame = x->m[i].p_curFrame;
			x->m[i].RR[x->m[i].RRCount].top = x->m[i].p_top+x->glob_v;
			x->m[i].RR[x->m[i].RRCount].left = x->m[i].p_left+x->glob_h;
			x->m[i].RR[x->m[i].RRCount].right = x->m[i].p_left + ((*x->m[i].p_pic[frame])->bounds.right - (*x->m[i].p_pic[frame])->bounds.left)+x->glob_h+1;//209
donot go lightly into this forest
			x->m[i].RR[x->m[i].RRCount].bottom = x->m[i].p_top + ((*x->m[i].p_pic[frame])->bounds.bottom - (*x->m[i].p_pic[frame])->bounds.top)+x->glob_v;

			sr = x->m[i].RR[x->m[i].RRCount];
			if(x->m[i].RRCount < NUM_DR_NBX-1)
				x->m[i].RRCount += 1;
			else
				error("Ask MRN again");
			DirtyIntersects(x,sr,i);
			}
		//.....pics_int(x,x->m[i].p_curFrame,i);
		}

	//for(i=0;i<=x->count;i++)
		//if(x->m[i].cue == 1)
	pics_int(x,x->m[i].p_curFrame,i);
//	MoviesTask (x->MG[0].MM, MOVIE_TIME);
	for(i=0;i<=x->MGCount;i++)
		{
		if(x->MG[i].MMactive)
			MoviesTask (x->MG[i].MM, MOVIE_TIME);
			//FlashMenuBar(0L);
		}
```

FIG. 20

```
	pics_qfn(x);
hell:
	ExitCallback();
}


void DirtyIntersects(Pics *x,Rect r,int thisone)
{
	int i;
	Rect r1,r2;
	int frame;

	x->m[thisone].cue = 1;

	for(i=0;i<=x->count;i++)
		{
		if(x->m[i].visible == 0)
			continue;

		if(i==thisone)
		  continue;

		//post("vis %s",x->m[i].AliasName);

		frame = x->m[i].p_curFrame;
		r1.top = x->m[i].p_top+x->glob_v;
		r1.left = x->m[i].p_left+x->glob_h;
		r1.right = x->m[i].p_left + ((*x->m[i].p_pic[frame])->bounds.right - (*x->m[i].p_pic[frame])->bounds.left)+x->glob_h+1;//209 donot go lightly into this forest
		r1.bottom = x->m[i].p_top + ((*x->m[i].p_pic[frame])->bounds.bottom - (*x->m[i].p_pic[frame])->bounds.top)+x->glob_v;

		if(SectRect(&r,&r1,&r2)==TRUE)
			{
			if(x->m[i].dirty != 1)
				{
				x->m[i].dirty = 2;
				//post("dirty add %s",x->m[i].AliasName);
				x->m[i].RR[x->m[i].RRCount] = r2;///////r1 is entire curr guy r2;
				x->m[i].cue = 1; //cue it
				if(x->m[i].RRCount < NUM_DR_NBX-1)
					x->m[i].RRCount += 1;
				else
					error("ASK MRN");
				}
			}
```

FIG. 21

```
		}
}

void pics_whatframe(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i=0,ndx;
	EnterCallback();

	if(argv[i].a_type==A_SYM)
		{
		ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;

	outlet_int (x->m_intout, x->m[ndx].p_curFrame);

bagout:
	ExitCallback();
}


void pics_next(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i=0,ndx;
	EnterCallback();

	if(x->lock)
		{
		error("LOCKED QFN");
		goto bagout;
		}

	if(argv[i].a_type==A_SYM)
		{
		ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;

	if (x->m[ndx].p_curFrame>=x->m[ndx].p_numFrames-1)
```

FIG. 22

```
                        x->m[ndx].p_curFrame = x->m[ndx].p_startFrame;
        else
                        x->m[ndx].p_curFrame = x->m[ndx].p_curFrame+1;

        if(x->m[ndx].dirty != FIRST_X)// do not nuke undrawn dirty rect
                {
                x->m[ndx].dirty = 1;
        //      x->m[ndx].dirtyleft = x->m[ndx].p_left;
        //      x->m[ndx].dirtytop = x->m[ndx].p_top;
                }
bagout:
        ExitCallback();
}

void pics_prev(Pics *x, Symbol *s, short argc, Atom *argv)
{
        int i=0,ndx;
        EnterCallback();

        if(x->lock)
                {
                error("LOCKED QFN");
                goto bagout;
                }

        if(argv[i].a_type==A_SYM)
                {
                ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
                if(ndx == -1)
                        goto bagout;
                }
        i++;

        if (x->m[ndx].p_curFrame<=0)
                        x->m[ndx].p_curFrame = x->m[ndx].p_numFrames-1;
        else
                        x->m[ndx].p_curFrame = x->m[ndx].p_curFrame-1;

        if(x->m[ndx].dirty != FIRST_X)// do not nuke undrawn dirty rect
                {
                x->m[ndx].dirty = 1;
        //      x->m[ndx].dirtyleft = x->m[ndx].p_left;
        //      x->m[ndx].dirtytop = x->m[ndx].p_top;
                }
```

FIG. 23

```
bagout:
	ExitCallback();
}

void pics_int(Pics *x, long n,int mp)
{
	EnterCallback();
	if(x->lock)
		{
		error("LOCKED QFN");
		goto bag;
		}
	//if (n >= x->m[mp].p_numFrames || n < x->m[mp].p_startFrame)
	//		goto bag;
	//x->m[mp].p_curFrame = n;
	//qelem_set(x->p_qelem);
bag:
	ExitCallback();
}

void pics_setxy(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i,ndx,t,l;
	EnterCallback();

	if(x->lock)
		{
		error("LOCKED QFN");
		goto bagout;
		}

	i=0;

	if(argv[i].a_type==A_SYM)
		{
		ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;
	if(argv[i].a_type==A_LONG)
		{
		l=x->m[ndx].p_left;
		x->m[ndx].p_left = argv[i].a_w.w_long;
```

FIG. 24

```
                if(x->Scale < 1.0)
                        x->m[ndx].p_left = x->m[ndx].p_left * x->Scale;
                if(x->m[ndx].lScale < 1.0)
                        x->m[ndx].p_left = x->m[ndx].p_left * x->m[ndx].lScale;

                }
        i++;
        if(argv[i].a_type==A_LONG)
                {
                t=x->m[ndx].p_top;
                x->m[ndx].p_top = argv[i].a_w.w_long;
                if(x->Scale < 1.0)
                        x->m[ndx].p_top = x->m[ndx].p_top * x->Scale;
                if(x->m[ndx].lScale < 1.0)
                        x->m[ndx].p_top = x->m[ndx].p_top * x->m[ndx].lScale;
                }
        i++;

        if(x->m[ndx].dirty != FIRST_X)// do not nuke undrawn dirty rect
                {
                x->m[ndx].dirty = 1;
        //      x->m[ndx].dirtyleft = l;
        //      x->m[ndx].dirtytop = t;
                }
bagout:
        ExitCallback();
}

void pics_winxy(Pics *x, Symbol *s, short argc, Atom *argv)
{
        int i;

        EnterCallback();

        i=0;
        if(argv[i].a_type==A_LONG)
                {
                x->wx = argv[i].a_w.w_long;
                }
        i++;
        if(argv[i].a_type==A_LONG)
                {
                x->wy = argv[i].a_w.w_long;
                }
```

FIG. 25

```
	i++;
	MoveWindow ((GrafPtr)&(x->Display->w_wind), x->wx, x->wy, FALSE);
bagout:
	ExitCallback();
}

void pics_movxy(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int ndx,i,wx=0,wy=0;
	Rect movieBox;

	EnterCallback();

	i=0;
	if(argv[i].a_type==A_SYM)
		{
		ndx=MGAlias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;
	if(argv[i].a_type==A_LONG)
		{
		wx = argv[i].a_w.w_long;
		}
	i++;
	if(argv[i].a_type==A_LONG)
		{
		wy = argv[i].a_w.w_long;
		}
	i++;

	GetMovieBox (x->MG[ndx].MM, &movieBox);
	SetRect (&movieBox, wx, wy,movieBox.right-movieBox.left,movieBox.bottom-movieBox.top);

	SetMovieBox (x->MG[ndx].MM, &movieBox);

bagout:
	ExitCallback();
}

void pics_BGxy(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i;
```

FIG. 26

```
	EnterCallback();

	i=0;
	if(argv[i].a_type==A_LONG)
		{
		x->BGx = argv[i].a_w.w_long;
		}
	i++;
	if(argv[i].a_type==A_LONG)
		{
		x->BGy = argv[i].a_w.w_long;
		}
	i++;

	x->draw_back = 1;
bagout:
	ExitCallback();
}

void pics_front(Pics *x, Symbol *s, short argc, Atom *argv)
{
	EnterCallback();
	BringToFront ((GrafPtr)&(x->Display->w_wind));
	ExitCallback();
}


void pics_setmovieframe(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i,frame,ndx;
	TimeValue tb;

	EnterCallback();

	i=0;
	if(argv[i].a_type==A_SYM)
		{
		ndx=MGAlias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;
	if(argv[i].a_type==A_LONG)
```

FIG. 27

```
                {
                frame = argv[i].a_w.w_long;
                }
        i++;

        tb = GetMovieTimeScale(x->MG[ndx].MM);

        SetMovieTimeValue(x->MG[ndx].MM,(tb * frame)/FRAME_PER_SEC);
        SetMovieRate(x->MG[ndx].MM,x->stopRate);

bagout:
        ExitCallback();

}

void pics_getmovie(Pics *x, Symbol *s, short argc, Atom *argv)
{
        Rect movieBox;
        TimeRecord tr;
        TimeValue start, end;


        int i,wx=0,wy=0;

        EnterCallback();

        i=0;

        x->MGCount += 1;

        if(x->MGCount > NUM_MOV)
                {
                error("Too Many Movies");
                goto bagout;
                }

        if(argv[i].a_type==A_SYM)
                {
                x->MG[x->MGCount].MM = GetMovie(x,argv[i].a_w.w_sym->s_name);
                }
        i++;

        if(argv[i].a_type==A_SYM)
                {
```

FIG. 28

```
                strcpy(x->MG[x->MGCount].MMAlias, argv[i].a_w.w_sym->s_name);
                }
        i++;

        if(argv[i].a_type==A_LONG)
                {
                wx = argv[i].a_w.w_long;
                }
        i++;
        if(argv[i].a_type==A_LONG)
                {
                wy = argv[i].a_w.w_long;
                }
        i++;

        GetMovieBox (x->MG[x->MGCount].MM, &movieBox);
        OffsetRect (&movieBox, wx, wy);

        //OffsetRect (&movieBox, -movieBox.left, -movieBox.top);
        SetMovieBox (x->MG[x->MGCount].MM, &movieBox);

        GoToBeginningOfMovie (x->MG[x->MGCount].MM);
        start = GetMovieTime(x->MG[x->MGCount].MM, &tr);

        GoToEndOfMovie (x->MG[x->MGCount].MM);
        end = GetMovieTime (x->MG[x->MGCount].MM, &tr);

        GoToBeginningOfMovie (x->MG[x->MGCount].MM);

        SetMovieGWorld(x->MG[x->MGCount].MM, (CGrafPtr)&(x->Display->w_wind), nil);

bagout:
        ExitCallback();
}


void pics_closeall(Pics *x, Symbol *s, short argc, Atom *argv)
{
        EnterCallback();
        HideWindow((struct GrafPort *)&(x->Display->w_wind));
        ExitCallback();
}

void pics_startmovie(Pics *x, Symbol *s, short argc, Atom *argv)
```

FIG. 29

```
{
	int i,ndx=0;

	EnterCallback();

	i=0;
	if(argv[i].a_type==A_SYM)
		{
		ndx=MGAlias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;

	x->MG[ndx].MMactive = 1;
	//GoToBeginningOfMovie (x->MG[ndx].MM);
	SetMovieRate(x->MG[ndx].MM,x->startRate);

	StartMovie(x->MG[ndx].MM);
bagout:
	ExitCallback();
}

void pics_stopmovie(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i,ndx=0;

	EnterCallback();

	i=0;
	if(argv[i].a_type==A_SYM)
		{
		ndx=MGAlias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;

	x->MG[ndx].MMactive = 0;
	SetMovieRate(x->MG[ndx].MM,x->stopRate);

	//GoToEndOfMovie (x->MG[ndx].MM);
bagout:
	ExitCallback();
```

FIG. 30

```
}
void pics_globxy(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i;
	long v,h
	EnterCallback();
	i=0;

	if(argv[i].a_type==A_LONG)
		{
		h = x->glob_h;
		x->glob_h = argv[i].a_w.w_long;
		}
	i++;
	if(argv[i].a_type==A_LONG)
		{
		v = x->glob_v;
		x->glob_v = argv[i].a_w.w_long;
		}
	i++;

	//dirty up for erase
	if(x->back_ground != -1)
		x->draw_back = 1;

bagout:
	ExitCallback();
}


void pics_hide(Pics *x, Symbol *s, short argc, Atom *argv)
{
	int i,ndx;
	EnterCallback();
	i=0;

	if(argv[i].a_type==A_SYM)
		{
		ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
		if(ndx == -1)
			goto bagout;
		}
	i++;
	x->m[ndx].visible = 0;
```

FIG. 31

```
bagout:
        ExitCallback();
}


void pics_show(Pics *x, Symbol *s, short argc, Atom *argv)
{
        int i,ndx;
        EnterCallback();
        i=0;

        if(argv[i].a_type==A_SYM)
                {
                ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
                if(ndx == -1)
                        goto bagout;
                }
        i++;
        x->m[ndx].visible = 1;
bagout:
        ExitCallback();
}

void pics_priority(Pics *x, Symbol *s, short argc, Atom *argv)
{
        int i,ndx;
        EnterCallback();
        i=0;

        if(argv[i].a_type==A_SYM)
                {
                ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
                if(ndx == -1)
                        goto bagout;
                }
        i++;
        if(argv[i].a_type==A_LONG)
                {
                x->m[ndx].p_pri = argv[i].a_w.w_long;
                if(x->m[ndx].p_pri>10)
                        {
                        x->m[ndx].p_pri = MAX_PRI;
                        post("Max Priority of %d substituted",MAX_PRI);
                        }
```

FIG. 32

```
                if(x->m[ndx].p_pri<0)
                        {
                        x->m[ndx].p_pri = 0;
                        post("Min Priority of %d substituted",0);
                        }
                }
        i++;

bagout:
        ExitCallback();
}

void sort_priorities(Pics *x)
{
        int i,p,ndx;

        ndx = 0;//reset for priority ordering
        for(p=0;p<=MAX_PRI;p++)
                {
                for(i=0;i<=x->count;i++)
                        {
                        if(x->m[i].p_pri==p)
                                {
                                x->prior[ndx]=i;
                                ndx++;
                                }
                        }
                }
}
void pics_setframe(Pics *x, Symbol *s, short argc, Atom *argv)
{
        int i,ndx;
        EnterCallback();
        i=0;

        if(x->lock)
                {
                error("LOCKED QFN");
                goto bagout;
                }

        if(argv[i].a_type==A_SYM)
                {
                ndx=Alias2Index(x,argv[i].a_w.w_sym->s_name);
```

FIG. 33

```
                if(ndx == -1)
                        goto bagout;
                }
        i++;
        if(argv[i].a_type==A_LONG)
                {
                x->m[ndx].p_curFrame = argv[i].a_w.w_long;
                if(x->m[ndx].p_curFrame < 0)
                        {
                        //error("setframe less than 0. 0 used.");
                        x->m[ndx].p_curFrame = 0;
                        }
                if(x->m[ndx].p_curFrame > x->m[ndx].p_numFrames-1)
                        {
                        //error("setframe to large for this seq. maximum used.");
                        x->m[ndx].p_curFrame = x->m[ndx].p_numFrames-1;
                        }
                }
        i++;

        if(x->m[ndx].dirty != FIRST_X)// do not nuke undrawn dirty rect
                {
                x->m[ndx].dirty = 1;
        //      x->m[ndx].dirtyleft = x->m[ndx].p_left;
        //      x->m[ndx].dirtytop = x->m[ndx].p_top;
                }

        if((strcmp(x->m[ndx].AliasName,"bg") == 0) ||
                        (strcmp(x->m[ndx].AliasName,"BG") == 0))
                        {
                        x->draw_back = 1;
                        for(i=0;i<=x->count;i++)
                                x->m[i].dirty = 1;
                        }

bagout:
        ExitCallback();
}

void pics_qfn(Pics *x)
{
        GrafPtr op,gp;
#if 0
        GWind *it;
```

FIG. 34

```
#endif
        Wind *it;
        short frame,bgframe;
        Rect nb,nbx;
        int i,j,k;
        int first = 0;
        Rect BIGGUYRect = {0,0,0,0};

        //return;
        EnterCallback();

        //if(Button())
        //      post(".....");
        if(x->lock)
                {
                error("MULTIPLE QFN CALLS");
                goto bag;
                }

        x->lock = 1;
        //PenSize(4,4);
#if 0
        it = gwind_get(x->p_sym);
        sort_priorities(x);
        if (!it ||!(op = gp = gwind_setport(it)))
                goto bag;
#endif
        it = x->Display;
        sort_priorities(x);
        if (!it ||!(op = gp = wind_setport(it)))
                goto bag;

        ////////////////////////
        // erase section
        if(x->draw_back)//draws entire back_grown use with caution
                {
                frame = x->m[x->back_ground].p_dFrame = x->m[x->back_ground].p_curFrame;

                nb.top = (*x->m[x->back_ground].p_pic[frame])->bounds.top;
                nb.left = (*x->m[x->back_ground].p_pic[frame])->bounds.left;
                nb.bottom = (*x->m[x->back_ground].p_pic[frame])->bounds.bottom;
                nb.right = (*x->m[x->back_ground].p_pic[frame])->bounds.right;

                BGCopy(x->the_off, x->m[x->back_ground].p_pic[frame], x->m[x->back_ground].p_mem[frame], &nb,x);
```

FIG. 35

```
		BIGGUYRect = nb;
		//.....flashes screen off_copyrect(x->the_off,&nb);
		//x->draw_back = 0;
		}
else
		{
		for(k=0;k<=x->count;k++)
			{
			if(x->m[k].dirty != 0)
				{
				int drc;

				for(drc=0;drc < x->m[k].RRCount;drc++)
					{
					nb = x->m[k].RR[drc];
					if(x->back_ground == -1)
						{
						EraseRect(&nb);
						}
					else
						{
						//EraseRect(&nb);
						bgframe = x->m[x->back_ground].p_curFrame;
						BGCopy(x->the_off, x->m[x->back_ground].p_pic[bgframe], x->m[x->back_ground].p_mem[bgframe], &nb,x);
						//ValidRect(&nb);
					//	InsetRect(&nb,-1,-1);
					//	FrameRect(&nb);
						}
					}
				}
			}//for
		}
// end erase section
//////////////////////////////
for(j=0;j<=x->count;j++)
		{
		i = x->prior[j];
		if(x->m[i].visible == 0)
			continue;
		frame = x->m[i].p_dFrame = x->m[i].p_curFrame;

		if (x->m[i].dirty != 0) //was if color qd
			{
			//post("Doing %s",x->m[i].AliasName);
```

FIG. 36

```
if (x->the_off)
        {
        if (x->m[i].p_pic && x->m[i].p_pic[frame])
                {
                nb = (*x->m[i].p_pic[frame])->bounds;
                OffsetRect(&nb,x->glob_h + x->m[i].p_left + x->m[i].p_offset[frame].h,
                                        x->glob_v + x->m[i].p_top + x->m[i].p_offset[frame].v);

                if(x->back_ground != -1)
                        {
                        int drc=0;
                        HLock(x->msk[i].runs);

                        //post("RR %d",x->m[i].RRCount);

                        for(drc=0;drc < x->m[i].RRCount;drc++)
                                {
                                nbx = x->m[i].RR[drc];
                                InsetRect(&nbx,-1,-1);
                                //if(x->m[i].dirty == 1)
                                //      nbx = nb;

                                if(PM5Copy(x->the_off,((long *)(*(x->msk[i].runs))) + frame * x->msk[i].RSize, x->m[i].p_pic[frame],x->m[x->back_ground].p_pic[x->m[x->back_ground].p_curFrame],
                                 x->m[i].p_mem[frame],x->m[x->back_ground].p_mem[x->m[x->back_ground].p_curFrame], &nb, &nbx)==1)
                                        {

                                        }
                                x->DirtyRects[x->NumDirty]=nbx;
                                if(x->NumDirty < NUM_DR-1)
                                        x->NumDirty+=1;
                                if(x->m[i].dirty == 2)
                                        ;//FrameRect(&nbx);//draw debugger
                                }
                        x->m[i].RRCount = 0;
                        HUnlock(x->msk[i].runs);
                        }

                //x->DirtyRects[x->NumDirty]=nb;
                //x->NumDirty+=1;
                //FrameRect(&nbx);//draw debugger
                }
        }
;//PrintRun(&x->msk[x->count]);
}//dirty
```

FIG. 37

```
            }// for x->count

{//////////////FINAL CLEANUP
int i;
//Rect xx;

//xx.top = 48;
//xx.left = 194;
//xx.right = 437;
//xx.bottom = 346;
//RGBColor xxcol;
//.BIGGUYRect.top = BIGGUYRect.left = 0;
//.BIGGUYRect.bottom = 400;
//.BIGGUYRect.right = 600;
//.x->draw_back = 1;
if(x->draw_back)
            {
            off_copyrect(x->the_off,&BIGGUYRect);
            x->draw_back = 0;
            }
else
            {
            for(i=0;i<x->NumDirty;i++)
                        {
                        off_copyrect(x->the_off,&x->DirtyRects[i]);
                        //off_copyrect(x->the_off,&xx);
                        //MWinBlit(x->the_off, &x->DirtyRects[i], x);

                        //FrameRect(&x->DirtyRects[i]);
                        }
            }
x->NumDirty = 0;

for(i=0;i<x->count;i++)
            {
            x->m[i].dirty = 0;//new turn off
            }
}//////////////
outlet_bang(x->m_bangout);/* send out a bang */

SetPort(op);

bag:
            x->lock = 0;
```

FIG. 38

```
	ExitCallback();
}

void pics_free(Pics *x)
{
	short i,j;

	EnterCallback();

	if(x->Display)
		HideWindow((struct GrafPort *)&(x->Display->w_wind));

	for(j=0;j<=x->count;j++)
		{
		//qelem_free(x->p_qelem);

		if(x->msk[j].runs)
			DisposeHandle(x->msk[j].runs);
		if (x->m[j].p_pic) {
			for (i=0; i < x->m[j].p_numFrames; i++) {
				if (x->m[j].p_pic[i])
					DisposePixMap(x->m[j].p_pic[i]);
			}
			DisposePtr((Ptr)x->m[j].p_pic);
		}
		if (x->m[j].p_mem) {
			for (i=0; i < x->m[j].p_numFrames; i++) {
				if (x->m[j].p_mem[i])
					DisposeHandle(x->m[j].p_mem[i]);
			}
			DisposePtr((Ptr)x->m[j].p_mem);
		}
		if (x->m[j].p_mask) {
			for (i=0; i < x->m[j].p_numFrames; i++) {
				if (x->m[j].p_mask[i])
					DisposeHandle((Handle)x->m[j].p_mask[i]);
			}
			DisposePtr((Ptr)x->m[j].p_mask);
		}
		if (x->m[j].p_offset)
			DisposePtr((Ptr)x->m[j].p_offset);
		}
	ExitCallback();
}
```

FIG. 39

```
/*
void pics_assist(Pics *x, void *b, long m, long a, char *s)
{
	EnterCallback();
	assist_string(3011,m,a,1,4,s);
	ExitCallback();
}
*/

void pics_assist (Pics *x, void *b, long msg, long arg, char *dst)
{
	EnterCallback();
		if (msg==ASSIST_INLET)
			{
			switch (arg)
				{
				case 0:
				case 1:
				case 2:
					strcpy(dst,"all messages to here");
				break;
				}
			}
		else if (msg==ASSIST_OUTLET)
			{
			switch (arg)
				{
				case 0:
					strcpy(dst,"bang when done drawing");
				break;
				case 1:
					strcpy(dst,"answer to what frame");
				break;
				}
			}
	ExitCallback();
}

void pics_test(Pics *x,long n)
{
	EnterCallback();
	if(n == 5)
		FlashMenuBar(0L);
	ExitCallback();
```

FIG. 40

```
}

void pics_bbang(Pics *x,long n)
{
	int i;
	int frame;
	Rect sr;
	EnterCallback();

//post("----------");
	for(i=0;i<=x->count;i++)
		{
		x->m[i].dirty = 1;
		}

	for(i=0;i<=x->count;i++)
		{
		if(x->m[i].visible == 0)
			continue;

		if(x->m[i].dirty == 1)//
			{
			//post("dirty %s",x->m[i].AliasName);

			frame = x->m[i].p_curFrame;
			x->m[i].RR[x->m[i].RRCount].top = x->m[i].p_top+x->glob_v;
			x->m[i].RR[x->m[i].RRCount].left = x->m[i].p_left+x->glob_h;
			x->m[i].RR[x->m[i].RRCount].right = x->m[i].p_left + ((*x->m[i].p_pic[frame])->bounds.right - (*x->m[i].p_pic[frame])->bounds.left)+x->glob_h+1;//209
donot go lightly into this forest
			x->m[i].RR[x->m[i].RRCount].bottom = x->m[i].p_top + ((*x->m[i].p_pic[frame])->bounds.bottom - (*x->m[i].p_pic[frame])->bounds.top)+x->glob_v;

			sr = x->m[i].RR[x->m[i].RRCount];
			if(x->m[i].RRCount < NUM_DR_NBX-1)
				x->m[i].RRCount += 1;
			else
				error("Ask MRN again");
			DirtyIntersects(x,sr,i);
			}
		//.....pics_int(x,x->m[i].p_curFrame,i);
		}

	//for(i=0;i<=x->count;i++)
		//if(x->m[i].cue == 1)
	pics_int(x,x->m[i].p_curFrame,i);
```

FIG. 41

```
//          MoviesTask (x->MG[0].MM, MOVIE_TIME);
            for(i=0;i<=x->MGCount;i++)
                        {
                        if(x->MG[i].MMactive)
                                    MoviesTask (x->MG[i].MM, MOVIE_TIME);
                                    //FlashMenuBar(0L);
                        }
            pics_qfn(x);
            ExitCallback();
}

void TestMethod(Pics *x, Symbol *s, short argc, Atom *argv)
{
            short i;

            EnterCallback();
            for (i=0; i < argc; i++)
                        {
                        switch (argv[i].a_type)
                                    {
                                    case A_LONG:
                                                post("argument %ld is a long: %ld", (long)i, argv[i].a_w.w_long);
                                    break;
                                    case A_SYM:
                                                post("argument %ld is a symbol: name %s", (long)i, argv[i].a_w.w_sym->s_name);
                                    break;
                                    case A_FLOAT:
                                                post("argument %ld is a float: %f", (long)i, argv[i].a_w.w_float);
                                    break;
                                    }
                        }
            ExitCallback();
}

void pics_Loadit(Pics *x, Symbol *s, short argc, Atom *argv)
{
            short i;
            short vol,refNum;
            char *fn;
            char name[128],maskname[128];
            short oldResFile;
            long        size = 0,sizesize = 0;
            Handle scratch;
```

FIG. 42

```
EnterCallback();

if(x->count >= NUM_PICS_FILES -1)
        {
        error("MAX PICS FIles reached");
        goto bag;
        }
i=0; //i < argc;
x->count += 1;
x->m[x->count].p_pic = 0;
x->m[x->count].p_offset = 0;
x->m[x->count].p_mem = 0;
x->m[x->count].p_top = x->m[x->count].p_left = 0;
x->m[x->count].p_mask = 0;
x->m[x->count].p_pri = 3;
x->m[x->count].p_curFrame = 0;
x->m[x->count].p_dFrame = 0;
x->m[x->count].p_startFrame = 0;
x->m[x->count].this_elem = x->count;
x->m[x->count].p_sym = x->p_sym;
x->m[x->count].visible=1;
x->m[x->count].xScale=x->Scale;
x->m[x->count].lScale=1.0;
x->m[x->count].angle=0.0;
x->m[x->count].dirty = FIRST_X;
x->m[x->count].RRCount = 0;


oldResFile = -3300;
//////x->m[x->count].p_backmode = !(sel==ps_pics);

if(argv[i].a_type == A_FLOAT)
        {
        x->m[x->count].lScale = argv[i].a_w.w_float;
        i++;argc--;
        }

if(argv[i].a_type == A_FLOAT)
        {
        if(argv[i].a_w.w_float < -0.001)
                x->m[x->count].angle = argv[i].a_w.w_float;//key flip/flop on neg angle
        else//convert to radians
                x->m[x->count].angle = (argv[i].a_w.w_float/360)*(2*3.14159);
        i++;argc--;
```

FIG. 43

```
        }

if(argv[i].a_type == A_SYM)
        {
        //post("argument %ld is a symbol: name %s", (long)i, argv[i].a_w.w_sym->s_name);

        fn = argv[i].a_w.w_sym->s_name;
        if (!locatefiletype(fn,&vol,'PICS',0L))
                {
                ctopcpy(name,fn);
                refNum = OpenRFPerm((Byte *)name,vol,0);
                if (refNum != -1)
                        {
                        oldResFile = CurResFile();
                        UseResFile(refNum);
                        inMask = 0;
                        if(loadit_pics_file(&(x->m[x->count]),fn) == -1)
                                goto ferr;
                        }
                }
        else
                {
                error("pics: no file %s",fn);
                goto bag;
                }
        }
i++;argc--;

if(argv[i].a_type == A_SYM)
        {
        //post("argument %ld is a symbol: name %s", (long)i, argv[i].a_w.w_sym->s_name);
        strcpy(x->m[x->count].AliasName, argv[i].a_w.w_sym->s_name);
        post("%s = %s", fn, x->m[x->count].AliasName);
        //-- special case for background
        if((strcmp(x->m[x->count].AliasName,"bg") == 0) ||
                (strcmp(x->m[x->count].AliasName,"BG") == 0))
                {
                x->back_ground = x->count;
                x->m[x->back_ground].visible = 0;
                x->draw_back = 1;
                }
        //--
        }
```

FIG. 44

```
#if 1     ////////////////////

          /////// MASK PART
          //x->count += 1; // no inc
          x->msk[x->count].p_pic = 0;
          x->msk[x->count].p_offset = 0;
          x->msk[x->count].p_mem = 0;
          x->msk[x->count].p_top = x->m[x->count].p_left = 0;
          x->msk[x->count].p_mask = 0;
          x->msk[x->count].p_pri = 3;
          x->msk[x->count].p_curFrame = 0;
          x->msk[x->count].p_dFrame = 0;
          x->msk[x->count].p_startFrame = 0;
          x->msk[x->count].this_elem = x->count;
          x->msk[x->count].p_sym = x->p_sym;
          x->msk[x->count].visible=0;//masks are invisible
          x->msk[x->count].runs = 0L;
          x->msk[x->count].xScale=x->Scale;
          x->msk[x->count].lScale=x->m[x->count].lScale;
          x->msk[x->count].angle=x->m[x->count].angle;

          // add msk to name
          //fn = argv->a_w.w_sym->s_name + "mask";
          //post("DEBUG name %s",name);

          ctopcpy(maskname,fn);

//        post("DEBUG maskname %s",maskname);

          maskname[maskname[0]+1]='-';
          maskname[maskname[0]+2]='m';
          maskname[maskname[0]+3]=0;
          maskname[0] += 2;

          post("DEBUG maskname %s",&(maskname[1]));


          if (!locatefiletype(&(maskname[1]),&vol,'PICS',0L))
                    {
                    refNum = OpenRFPerm((Byte *)maskname,vol,0);
                    if (refNum != -1)
                              {
                              oldResFile = CurResFile();
                              UseResFile(refNum);
```

FIG. 45

```
                        inMask=1;
                        if(loadit_pics_file(&(x->msk[x->count]),"mask part") == -1)
                                goto ferr;
                        }
                }
        else
                {
                // report no mask file
                error("pics: no mask file %s",fn);
                goto bag;
                }
        {// encode the mask
        int ix;
        x->msk[x->count].runs = NewHandleClear(RUN_ELEMS * 4 * x->msk[x->count].p_numFrames);

        // encode and free masks just after they are read
        for(ix=0;ix<x->msk[x->count].p_numFrames;ix++)
                {
                //PixMapHandle pmc;
                HLock(x->msk[x->count].runs);
                //pmc = (PixMapHandle) NewHandleClear(GetHandleSize((Handle)x->msk[x->count].p_pic[ix]));
                //HLock((Handle)pmc);
                //CopyPixMap(x->msk[x->count].p_pic[ix],pmc);
                if(MEncode(x->the_off,((long *)(*(x->msk[x->count].runs))) + ix * RUN_ELEMS,x->msk[x->count].p_pic[ix],x->msk[x->count].p_mem[ix])==1)
                        ;//PrintRun(&x->msk[x->count]);
                //MEncode(x->the_off,((long *)(*(x->msk[x->count].runs))) + ix * RUN_ELEMS,pmc,x->msk[x->count].p_mem[ix]);
//PrintRun(&x->msk[x->count]);
                pics_free_pic_inmask(x,x->count,ix);
                HUnlock(x->msk[x->count].runs);
                //HUnlock((Handle)pmc);
                //DisposeHandle((Handle)pmc);
                }
        //get max size of a run
        for(ix=0;ix<x->msk[x->count].p_numFrames;ix++)
                {

                size = GetRunsSize(((long *)(*(x->msk[x->count].runs))) + ix * RUN_ELEMS);
                //post("size %d",size);
                if(size > sizesize)
                        sizesize = size;
                }
        x->msk[x->count].RSize = sizesize;

        scratch = x->msk[x->count].runs;
```

FIG. 46

```
		x->msk[x->count].runs = NewHandleClear(x->msk[x->count].RSize * 4 * x->msk[x->count].p_numFrames);

		//copy old into new
#if 1
		HLock(scratch);
		HLock(x->msk[x->count].runs);
		for(ix=0;ix<x->msk[x->count].p_numFrames;ix++)
			{
			long *src,*dst;

			dst = ((long *)(*(x->msk[x->count].runs))) + ix * x->msk[x->count].RSize;
			src = ((long *)(*(scratch))) + ix * RUN_ELEMS;

			CopyRun(src,dst);

			//dst = ((long *)(*(x->msk[x->count].runs))) + ix * x->msk[x->count].RSize;
			//src = ((long *)(*(scratch))) + ix * RUN_ELEMS;

			//CMPRun(src,dst);

			}
#endif
		//PrintRun(&x->msk[x->count]);

		HUnlock(x->msk[x->count].runs);
		HUnlock(scratch);
		//delete old handle,make sure its unlocked
		DisposeHandle(scratch);

		pics_free_misc(x,x->count);

		//PrintRun(&x->msk[x->count]);

		}
		/////// end mask part
#endif	//////////////////////

	if (oldResFile != -3300)
		{
		CloseResFile(refNum);
		UseResFile(oldResFile);
		}
	ExitCallback();
	return;
```

FIG. 47

```
ferr:
        if (oldResFile != -3300)
                {
                CloseResFile(refNum);
                UseResFile(oldResFile);
                }
bag:
        //freeobject((Object *)x);
        ExitCallback();
        return;
}

int loadit_pics_file(picsGuts *x,char *name)
{
                Rect frameResult;
                short np;
                register short i,res;
                PicHandle thePic;

                EnterCallback();
                np = Count1Resources('PICT');

                if (!np) {
                        //error("pics: no PICT resources in %s",fn);
                        error("pics: no PICT resources in this file");
                        ExitCallback();
                        return(-1);
                }
                if (hasColorQD) {
                        post("pics: reading file... %s",name);
                        x->p_pic = (PixMapHandle *)NewPtr(4L * np);  /* getbytes? */
                        x->p_mem = (Handle *)NewPtr(4L * np);
                        x->p_mask = (BitMapHandle *)NewPtr(4L * np);
                        x->p_offset = (Point *)NewPtr(4L * np);
                        for (i=0; i < np; i++) {
                                x->p_pic[i] = 0;
                                x->p_mem[i] = 0;
                                x->p_mask[i] = 0;
                                x->p_offset[i].h = x->p_offset[i].v = 0;
                        }
                        x->p_numFrames = np;
                        for (i=0; i < np; i++) {
                                thePic = (PicHandle)Get1Resource('PICT',128+i);
                                if (!thePic || ResError() || MemError()) {
```

FIG. 48

```
                                        error("pics: difficulty reading file");
                                        ExitCallback();
                                        return(-1);
                                }
                                x->p_pic[i] = NewPixMap();
                                x->p_mem[i] = newhandle(0L);
                                x->p_mask[i] = (BitMapHandle)newhandle(0L);
                                res = DrawCPicInto(thePic,x->p_pic[i],x->p_mem[i],x->p_mask[i],0,
                                                &frameResult,x->p_offset+i,x);

                                post("Frame %d size %ld",i,GetHandleSize(x->p_mem[i]),res);
                                ReleaseResource((Handle)thePic);
                                if (i==0)
                                        x->p_bounds = frameResult;
                        }
                } else
                        post("pics: requires color quickdraw");
                ExitCallback();
                return 0;
}

void *pics_new(Symbol *sel, short argc, Atom *argv)
{
        Pics *x;
        //GWind *fooG;

        EnterCallback();
        x = (Pics *)newobject(pics_class);
        x->count = -1; //inc on loadit
        //x->p_qelem = qelem_new(x,(method)pics_qfn);
        x->m_intout = intout(x);
        x->m_bangout = bangout(x);
        x->glob_h = x->glob_v = 0;
        x->back_ground = -1;
        x->draw_back = 0;
        x->NumDirty = 0;
        x->lock = 0;
        x->m = (struct picsguts *)NewPtrClear(sizeof(picsGuts) * NUM_PICS_FILES);
        x->msk = (struct picsguts *)NewPtrClear(sizeof(picsGuts) * NUM_PICS_FILES);


        // ---------
        // this sets up the window association
        if (argc && argv->a_type==A_SYM)
```

FIG. 49

```
        {
        x->p_sym = argv->a_w.w_sym; // this is actualy the graphics window
        argv++;
        argc--;
        }
else
        {
        x->p_sym = 0;
        }


if (argc && argv->a_type == A_LONG)
        {
        x->wtop = argv->a_w.w_long;
        argc--;
        argv++;
        }
if (argc && argv->a_type == A_LONG)
        {
        x->wleft = argv->a_w.w_long;
        argc--;
        argv++;
        }
if (argc && argv->a_type == A_LONG)
        {
        x->wbottom = argv->a_w.w_long;
        argc--;
        argv++;
        }
if (argc && argv->a_type == A_LONG)
        {
        x->wright = argv->a_w.w_long;
        argc--;
        argv++;
        }
// init in case they forget
x->Scale = 1.0;
if (argc && argv->a_type == A_FLOAT)
        {
        x->Scale = argv->a_w.w_float;
        //xScale = .5;
        argc--;
        argv++;
        }
```

FIG. 50

```
        x->Display = wind_new (x, x->wtop,x->wleft,x->wright,x->wbottom,0x00f3);
        wind_settitle(x->Display,x->p_sym->s_name,0);
        x->the_off = off_new(&x->Display->w_wind.port);
        x->Movied = 0;
        x->MGCount = -1;
        x->stopRate = FixRatio(0,1);
        x->startRate = FixRatio(1,1);
        x->BGx = x->BGy = 0;

        // ----------



        ExitCallback();
        return (x);

}

/* ----- convert a picture into a bitmap ---------- */
// called:
//res = DrawCPicInto(thePic,x->p_pic[i],x->p_mem[i],x->p_mask[i],0,
//                                                    &frameResult,x->p_offset+i,x);
//
short DrawCPicInto(PicHandle ph, PixMapHandle pm, Handle mem, BitMapHandle mask, short slop,
        Rect *frameResult, Point *offset,picsGuts *x)
{
        CGrafPort tempPort,tempPort2;
        GrafPtr savePort;
        Rect gTemp;
        short len,height,rowBytes,theDepth;
        long theSize;

        Handle Tmem;

        GetPort(&savePort);
        OpenCPort(&tempPort);
        OpenCPort(&tempPort2);
        theDepth = (*(tempPort.portPixMap))->pixelSize;
        *frameResult = (*ph)->picFrame;
        //scale frame result here
        if(x->xScale < 1.0)
                frameResult->right *= x->xScale;
        if(x->lScale < 1.0)
```

FIG. 51

```
        frameResult->right *= x->lScale;

if(x->xScale < 1.0)
        frameResult->bottom *= x->xScale;
if(x->lScale < 1.0)
        frameResult->bottom *= x->lScale;

len = frameResult->right - frameResult->left;
height = frameResult->bottom - frameResult->top;
//rowBytes = (short)((((long)(((long)theDepth * (len)) + 15)) / 16L) * 2L);
rowBytes = (short)((((long)(((long)theDepth * (len)) + 31)) / 32L) * 4L);

theSize = (long)rowBytes * (long)height;
SetHandleSize(mem,theSize);
if (MemError())
        return (FALSE);

Tmem = NewHandle(0L);
SetHandleSize(Tmem,theSize);
if (MemError())
        return (FALSE);

SetRect(&gTemp,0,0,len,height);
(*(tempPort.portPixMap))->rowBytes = rowBytes + 32768;  /* set high bit for some reason */
(*(tempPort2.portPixMap))->rowBytes = rowBytes + 32768;  /* set high bit for some reason */

SetRect(&((*(tempPort.portPixMap))->bounds),0,0,len,height);
SetRect(&((*(tempPort2.portPixMap))->bounds),0,0,len,height);

HLock(mem);
HLock(Tmem);

HLock((Handle)tempPort.portPixMap);
HLock((Handle)tempPort2.portPixMap);

CalcBaseAddr(tempPort.portPixMap,mem);
CalcBaseAddr(tempPort2.portPixMap,Tmem);


offset->h = (*ph)->picFrame.left;
offset->v = (*ph)->picFrame.top;
OffsetRect(frameResult,-((*ph)->picFrame.left)-slop,
        -((*ph)->picFrame.top)-slop);
```

FIG. 52

```
BlockMove(frameResult,&(*pm)->bounds,8L);

SetPort((GrafPtr)&tempPort);

EraseRect(frameResult);
EraseRect(&gTemp);
//FillRect(frameResult,&qd.black);
//FillRect(&gTemp,&qd.black);
///////DrawPicture(ph,frameResult);
CopyPixMap(tempPort.portPixMap,pm);
CopyPixMap(tempPort2.portPixMap,pm);

if(x->angle > 0.001)
        {
        SetPort((GrafPtr)&tempPort2);
        EraseRect(frameResult);
        EraseRect(&gTemp);
        //FillRect(frameResult,&qd.black);
        //FillRect(&gTemp,&qd.black);

        DrawPicture(ph,frameResult);
        //FrameRect(&vr);
        //FlashMenuBar(0L);

        MRotate(tempPort.portPixMap,tempPort2.portPixMap,Tmem,x->angle);

        }
else if(x->angle < -359.0)
        {
        SetPort((GrafPtr)&tempPort); //diff temp port
        EraseRect(frameResult);
        EraseRect(&gTemp);
        DrawPicture(ph,frameResult);
        MFlip(tempPort2.portPixMap,tempPort.portPixMap,mem,x->angle);
        MFlop(tempPort.portPixMap,tempPort2.portPixMap,Tmem,x->angle);
//      CopyPixMap(tempPort2.portPixMap,tempPort.portPixMap);


        }
else if(x->angle < -179.0)
        {
        SetPort((GrafPtr)&tempPort2);
        EraseRect(frameResult);
        EraseRect(&gTemp);
```

FIG. 53

```
                DrawPicture(ph,frameResult);
                MFlip(tempPort.portPixMap,tempPort2.portPixMap,Tmem,x->angle);
                }
        else if(x->angle < -0.001)
                {
                SetPort((GrafPtr)&tempPort2);
                EraseRect(frameResult);
                EraseRect(&gTemp);
                DrawPicture(ph,frameResult);
                MFlop(tempPort.portPixMap,tempPort2.portPixMap,Tmem,x->angle);
                }
        else
                {
                SetPort((GrafPtr)&tempPort);
                EraseRect(frameResult);
                EraseRect(&gTemp);
                //FillRect(frameResult,&qd.black);
                //FillRect(&gTemp,&qd.black);

                DrawPicture(ph,frameResult);
                }

        HUnlock((Handle)tempPort.portPixMap);
        CloseCPort(&tempPort);

        HUnlock((Handle)tempPort2.portPixMap);
        CloseCPort(&tempPort2);

        HUnlock(mem);
        HUnlock(Tmem);
        DisposeHandle(Tmem);
        SetPort(savePort);
        return (TRUE);
}


void CalcBaseAddr(PixMapHandle ph, Handle bh)
{
        (*ph)->baseAddr = (Ptr)*bh;
}
```

FIG. 54

```
int PM5Copy(void *off,long *r, PixMapHandle map, PixMapHandle backmap, Handle mem,Handle backmem, Rect *dstR, Rect *clipR)
{
        //int i,j;
        int w,wf,w2;
#ifdef BIT_16
        short *f,*BackGround,*WindBase;
#else
        long *f,*BackGround,*WindBase;
        long *wb_p, *f_p,*bg_p;
#endif
        GWorldPtr wo;
        OffScreen *offs;
        Point     pt;
        int row_w;
        long c=0;
        int NumLines = 0;
        int WinRight, WinBot, WinLeft, WinTop;


        Rect *bounds = &(*map)->bounds;

#define ROW_DIV  PIXEL_SIZE

        HLock(mem);
        HLock(backmem);
        HLock((Handle)map);
        HLock((Handle)backmap);

        CalcBaseAddr(map,mem);
        CalcBaseAddr(backmap,backmem);


        offs =(OffScreen *)off;
        //w2 = (*(offs->offPix))->rowBytes; the evil line of crashes
        //w2 = offs->offBits.rowBytes;
        wo = offs->world;
#ifdef BIT_16
        WindBase = (short *)GetPixBaseAddr(wo->portPixMap);
#else
        WindBase = (long *)GetPixBaseAddr(wo->portPixMap);
#endif
        w = (*map)->rowBytes & 0x3fff;
        wf = (*backmap)->rowBytes & 0x3fff;
```

FIG. 55

```
        w2 = ((*(wo->portPixMap))->rowBytes & 0x3fff);

        if(LockPixels (wo->portPixMap))
                ;
        else
                error("pics : background has been purged");
WinRight = clipR->right;
WinBot =  clipR->bottom;
WinLeft = clipR->left;
WinTop = clipR->top;
        //f = (short *)GetPixBaseAddr(map->portPixMap);
#ifdef BIT_16
        f = (short *)(*map)->baseAddr;
        BackGround = (short *)(*backmap)->baseAddr;
#else
        f = (long *)(*map)->baseAddr;
        BackGround = (long *)(*backmap)->baseAddr;
#endif

        pt.h = dstR->left;
        pt.v = dstR->top;

        //akey = (short *) WindBase;
        WindBase += pt.h;
        WindBase += ((*(wo->portPixMap))->rowBytes & 0x3fff) * pt.v / ROW_DIV;

        BackGround += pt.h;
        BackGround += (wf) * pt.v / ROW_DIV;

                wb_p = WindBase;
                f_p = f;
                bg_p = BackGround;

        row_w = (dstR->right - dstR->left);

        //for(j=0;j<dstR->bottom - dstR->top;j++)
                {
                //for(i=0;i<row_w;i++)
                while(r[c] != STP_TAG)
                        {
                        if(c>RUN_ELEMS)
                                {
                                error("Something is wrong, ask Mark about it");
                                return 1;
```

FIG. 56

```
        }
//if((pt.v + NumLines < WinBot) && (pt.v + NumLines > WinTop) ) // vert clip
        {
        switch(r[c])
                {
                case KEY_TAG:
                        WindBase += r[c+1];
                        f += r[c+1];
                        BackGround += r[c+1];
                        //c+=2;
                        break;

                case PIX_TAG:
                        {
                        int d,s;
                        s = r[c+1];
                        d=0;
                /* put this optimization back in some time
                        for(;d<s-4;d+=4)
                                {
                                //if(d==0)
                                if((pt.v + NumLines > WinBot) && (pt.v + NumLines < WinTop) ) // vert clip
                                        {
                                        break;
                                        }
                                if(((pt.h) + (WindBase - wb_p) > WinRight - 4) && ((pt.h) + (WindBase - wb_p) > WinLeft))
                                        {
                                        break;
                                        }
                                else
                                        {
                                *WindBase++ = *f++;
                                *WindBase++ = *f++;
                                *WindBase++ = *f++;
                                *WindBase++ = *f++;
                                BackGround += 4;
                                        }
                                }
                        */
                        for(;d<s;d++)
                                {
                                if((pt.v + NumLines > WinBot)) // vert clip
                                        {
                                        break;
```

FIG. 57

```
                    }
                if(((pt.h) + (WindBase - wb_p) > WinRight) || ((pt.h) + (WindBase - wb_p) < WinLeft))
                    {
                    //FlashMenuBar(0L);
                    WindBase++;
                    BackGround++;
                    f++;
                    }
                else if((pt.v + NumLines > WinTop))
                    {
                //if(d==0)
                *WindBase = *f;
                //*WindBase = 0x0000ff;
                WindBase += 1;
                f += 1;
                BackGround += 1;
                    }
                }
        //c+=2;
        }
        break;
case CMP_TAG:
        {
        int s;
        s = r[c+1];
        if(((pt.h) + (WindBase - wb_p) < WinRight) && ((pt.h) + (WindBase - wb_p) > WinLeft))
        if(((pt.v) + (NumLines) < WinBot) && ((pt.v) + (NumLines) > WinTop))  //was 600 very reciently

                {
                long degree = (s) & 0xff;// reverse ramp
                long BackPix = *WindBase;    //*BackGround
                long Pix = *f; //this is non aliased pixel needs to be fixed in source image

                *WindBase = ((                  (RED(BackPix)*(255-degree)  + RED(Pix)*(degree))
/255    )<<16) |
                                        ((              (GREEN(BackPix)*(255-degree)  + GREEN(Pix)*(degree))
/255            )<<8) |
                                        (               (BLUE(BackPix)*(255-degree)+ BLUE(Pix)*(degree))
        /255/* << 0 */);
                }
        WindBase += 1;
        f += 1;
        BackGround += 1;
        }
```

FIG. 58

```
                                        break;
                                case END_TAG:
                                        {
                                        f    = f_p += (w/ROW_DIV);
                                        BackGround    = bg_p += (wf/ROW_DIV);
                                        WindBase = wb_p += (w2/ROW_DIV);
                                        NumLines += 1;
                                        //c += 2;
                                        }
                                        break;
                                case STP_TAG:
                                        {

                                        }
                                        break;
                                }
                        }//if for vert clip
                c+=2;
                }//while c

        }
        ////....ValidRect(bounds);
        /*
        *WindBase++ = 0xff00ff;
        *WindBase++ = 0xff00ff;
        *WindBase++ = 0xff00ff;
        *WindBase++ = 0xff00ff;
        *WindBase++ = 0xff00ff;
        */
        UnlockPixels (wo->portPixMap);

        HUnlock(mem);
        HUnlock(backmem);
        HUnlock((Handle)map);
        HUnlock((Handle)backmap);

        return 0;
}


void pics_update (Pics *x)
{
        GrafPtr op,gp;
        Wind *it;
```

FIG. 59

```
        EnterCallback();

        it = x->Display;

        if (!it ||!(op = gp = wind_setport(it)))
                goto bag;

//      InvalRect(&x->Display->w_wind.port.portRect);

        BeginUpdate((GrafPtr)&(x->Display->w_wind));

        if(x->back_ground != -1)
                x->draw_back = 1;

        ValidRect(&x->Display->w_wind.port.portRect);
        //for(i=0;i<x->count;i++)
        //      x->m[i].dirty=1;

        //pics_qfn(x);


        EndUpdate((GrafPtr)&(x->Display->w_wind));

        SetPort(op);
bag:
        ExitCallback();
}


void pics_activate (Pics *x, short active)
{
EnterCallback();
//if(x->back_ground != -1)
//      x->draw_back = 1;
//pics_bang(x);
ExitCallback();
}


void pics_close (Pics *x)
{
EnterCallback();

HideWindow((struct GrafPort *)&(x->Display->w_wind));
```

FIG. 60

```
pics_free(x);
ExitCallback();

}

int MEncode(void *off,long *r,PixMapHandle maskmap,Handle maskmem)
{
        int i,j;
        int maskrowB;
        long        *rPtr;
        long *Mask,*MaskStart,*ms;
        int row_w;

#define ROW_DIV  PIXEL_SIZE

        HLock(maskmem);
        HLock((Handle)maskmap);
        CalcBaseAddr(maskmap,maskmem);

        maskrowB = (*maskmap)->rowBytes & 0x3fff;

        MaskStart = Mask = (long *)(*maskmap)->baseAddr;

        row_w = (*maskmap)->bounds.right - (*maskmap)->bounds.left;
        rPtr = r; //set it
        //post("toop bot = %d",(*maskmap)->bounds.bottom - (*maskmap)->bounds.top);
        //post("left right = %d",(*maskmap)->bounds.right - (*maskmap)->bounds.left);

        for(j=0;j< (*maskmap)->bounds.bottom - (*maskmap)->bounds.top;j++)
                {
                ms = Mask;
                for(i=0;i< (*maskmap)->bounds.right - (*maskmap)->bounds.left;)//no i++ big change
                        {
                        switch(*Mask)
                        {
                                case KEY_COL:
                                        *rPtr=KEY_TAG;
                                        rPtr++;
                                        *rPtr = 1;
                                        Mask++;
                                        while(*Mask == KEY_COL)
                                                {
                                                if( i >= (*maskmap)->bounds.right - (*maskmap)->bounds.left)
                                                        {
```

FIG. 61

```
                        //          post("breaking");
                                    break;
                                    }
                        *rPtr+=1;
                        Mask++;
                        }
                i+= *rPtr;//-1;
                rPtr += 1;
                break;
        case WHITE_COL:
                //*WindBase = *f;
                *rPtr=PIX_TAG;
                rPtr++;
                *rPtr = 1;
                Mask++;
                while(*Mask == WHITE_COL)
                        {
                        *rPtr+=1;
                        Mask++;
                        i++;
                        if(i>= (*maskmap)->bounds.right - (*maskmap)->bounds.left)
                                break;
                        }
                //i+= *rPtr;//-1;
                rPtr += 1;
                break;
        //else //---if((*Mask != KEY_COL) && (*Mask != BLACK_COL))//Alias background with mask here
        default:
                *rPtr=CMP_TAG;
                rPtr++;
                *rPtr = *Mask;
                Mask++;
                //while(*Mask != KEY_COL)
                //          {
                //          *rPtr+=1;
                //          Mask++;
                //          }
                i+= 1;//*rPtr-1;
                rPtr += 1;
                break;
        }//switch
if((long *)rPtr-(long *)r > RUN_ELEMS-2)
        {
        FlashMenuBar(0L);
```

FIG. 62

```
                                post("i=%d rPtr=%d",i,(long *)rPtr-(long *)r);
                                error("image is too complex for simplification, repent, the end is near");
                                goto hell;
                                }
                        //Mask += 1;
                        }//for i
                *rPtr=END_TAG;
                rPtr += 1;
                *rPtr=209;
                rPtr += 1;
                Mask    = ms + ((maskrowB/ROW_DIV));
                }
hell:
        *rPtr=STP_TAG;
        rPtr += 1;
        *rPtr=219;
        rPtr += 1;
        HUnlock(maskmem);
        HUnlock((Handle)maskmap);

        return 0;
}



void PrintRun(picsGuts *pg)
{
        int i;
        long *r;

        HLock(pg->runs);
        r = (long *)*(pg->runs);

        for(i=0;r[i] != STP_TAG;i+=2)
                {
                if(r[i] == KEY_TAG)
                        post("KEY TAG = %d", r[i+1]);
                if(r[i] == PIX_TAG)
                        post("PIX TAG = %d", r[i+1]);
                if(r[i] == CMP_TAG)
                        post("CMP TAG = %d", r[i+1]);
                if(r[i] == END_TAG)
                        post("END TAG = %d", r[i+1]);
```

FIG. 63

```
                if(r[i] == STP_TAG)
                        post("STP TAG = %d", r[i+1]);
                }
        HUnlock(pg->runs);

}


void SpotRun(picsGuts *pg,long *spot)
{
        int i;
        long *r;

        HLock(pg->runs);
        r = (long *)*(pg->runs);


        for(i=0;r[i] != STP_TAG;i++)
                {
                if(r[i] == KEY_TAG)
                        {
                        int x=r[i+1];
                        while(x>0)
                                {
                                *spot=0xff0000;
                                spot+=1;
                                x-=1;
                                }

                        //post("KEY TAG = %d", r[i+1]);
                        }
                if(r[i] == PIX_TAG)
                        {
                        int x=r[i+1];
                        while(x>0)
                                {
                                *spot=0x00ff00;
                                spot+=1;
                                x-=1;
                                }
                        //post("PIX TAG = %d", r[i+1]);
                        }
                if(r[i] == CMP_TAG)
                        {
```

FIG. 64

```
                        int x=r[i+1];
                        while(x>0)
                                {
                                *spot=0x0000ff;
                                spot+=1;
                                x-=1;
                                }
                        //post("CMP TAG = %d", r[i+1]);
                        }
                if(r[i] == END_TAG)
                        {
                        int x=r[i+1];
                        while(0)
                                {
                                //*spot++=0xff0000;
                                }
                        //post("END TAG = %d", r[i+1]);
                        }
                if(r[i] == STP_TAG)
                        {
                        int x=r[i+1];
                        while(0)
                                {
                                //*spot++=0xff0000;
                                }
                        //post("STP TAG = %d", r[i+1]);
                        }
                }
        HUnlock(pg->runs);

}


void pics_free_masks(Pics *x,int j)
{
        short i;

        EnterCallback();

        //for(j=0;j<=x->count;j++)
                {

                if (x->msk[j].p_pic) {
```

FIG. 65

```
                for (i=0; i < x->msk[j].p_numFrames; i++) {
                        if (x->msk[j].p_pic[i])
                                DisposePixMap(x->msk[j].p_pic[i]);
                }
                DisposePtr((Ptr)x->msk[j].p_pic);
        }
        if (x->msk[j].p_mem) {
                for (i=0; i < x->msk[j].p_numFrames; i++) {
                        if (x->msk[j].p_mem[i])
                                DisposeHandle(x->msk[j].p_mem[i]);
                }
                DisposePtr((Ptr)x->msk[j].p_mem);
        }
        if (x->msk[j].p_mask) {
                for (i=0; i < x->msk[j].p_numFrames; i++) {
                        if (x->msk[j].p_mask[i])
                                DisposeHandle((Handle)x->msk[j].p_mask[i]);
                }
                DisposePtr((Ptr)x->msk[j].p_mask);
        }
        if (x->msk[j].p_offset)
                DisposePtr((Ptr)x->msk[j].p_offset);
        }
    ExitCallback();
}


void pics_free_misc(Pics *x,int j)
{
    EnterCallback();

        if (x->msk[j].p_pic)
                {
                DisposePtr((Ptr)x->msk[j].p_pic);
                }
        if (x->msk[j].p_mem)
                {
                DisposePtr((Ptr)x->msk[j].p_mem);
                }
        if (x->msk[j].p_mask)
                {
                DisposePtr((Ptr)x->msk[j].p_mask);
```

FIG. 66

```
                }
        if (x->msk[j].p_offset)
                {
                DisposePtr((Ptr)x->msk[j].p_offset);
                }

        ExitCallback();
}

void pics_free_pic_inmask(Pics *x,int j,int i)
{
        EnterCallback();

                if (x->msk[j].p_pic)
                        {
                        if (x->msk[j].p_pic[i])
                                DisposePixMap(x->msk[j].p_pic[i]);
                        }
                if (x->msk[j].p_mem)
                        {
                        if (x->msk[j].p_mem[i])
                                DisposeHandle(x->msk[j].p_mem[i]);
                        }
                if (x->msk[j].p_mask)
                        {
                        if (x->msk[j].p_mask[i])
                                DisposeHandle((Handle)x->msk[j].p_mask[i]);
                        }

        ExitCallback();
}

long GetRunsSize(long *r)
{
        long c=0;

        while(r[c] != STP_TAG)
                c++;

        return c+2;//2
}

void CopyRun(long *src,long *dst)
{
```

FIG. 67

```
	long c=0;

	while(src[c] != STP_TAG)
		{
		dst[c] = src[c];
		c++;
		}
	dst[c] = STP_TAG;
	dst[c+1] = 219;

}

void CMPRun(long *src,long *dst)
{
	long c=0;

	while(src[c] != STP_TAG)
		{
		if(dst[c] != src[c])
			FlashMenuBar(0L);
		c++;
		}
}

////////////////////////////
//Movie support
////////////////////////////

Movie	GetMovie (Pics *x,char *mmname)
{
	OSErr				err;
	SFTypeList			typeList = {MovieFileType, 0,0,0};
	//StandardFileReply reply;
	Movie				aMovie = nil;
	short				movieResFile;
	FSSpec				spec;
	short		vol,bin;
	char pmmname[128];


	if(x->Movied == 0)
		{
		err = EnterMovies();
		if(err)
```

FIG. 68

```
                        {
                        error("cant enter movies");
                        return nil;
                        }
                x->Movied = 1;
                }
        //StandardGetFilePreview (nil, 1, typeList, &reply);
        locatefile (mmname, &vol, &bin);
        ctopcpy(pmmname,mmname);
        err = FSMakeFSSpec (vol, 0, (unsigned char *)pmmname, &spec);
        if(err != noErr)
                FlashMenuBar(0L);
        //if (reply.sfGood)
        if(err == noErr)
                {
                err = OpenMovieFile (&spec, &movieResFile, fsRdPerm);
                if (err == noErr)
                        {
                        short      movieResID = 0;
                        Str255     movieName;
                        Boolean    wasChanged;

                        err = NewMovieFromFile (&aMovie, movieResFile, &movieResID, movieName,
                                newMovieActive, &wasChanged);
                        CloseMovieFile (movieResFile);
                        }
                }
        return aMovie;
}

void MRotate(PixMapHandle idestmap,PixMapHandle imap,Handle imem,double angle)
{
        int i,j;
        int ImgrowB;
        long *Img,*ImgStart,*hs,*ds;
        long *Dest,*DestStart;

        int row_w;
        int centerx,centery;

        double s_ang,c_ang;

#define ROW_DIV  PIXEL_SIZE
```

FIG. 69

```
HLock(imem);
HLock((Handle)imap);
CalcBaseAddr(imap,imem);

s_ang = sin(angle);
c_ang = cos(angle);
ImgrowB = (*imap)->rowBytes & 0x3fff;

ImgStart = Img = (long *)(*imap)->baseAddr;
DestStart = Dest = (long *)(*idestmap)->baseAddr;

for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom);j+=1)
        for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right);i+=1)
                *(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = 0;//fill blac to rid rot zones

post("rot");

row_w = (*imap)->bounds.right - (*imap)->bounds.left;

centerx = ((*imap)->bounds.right - (*imap)->bounds.left)/2;
centery = ((*imap)->bounds.bottom - (*imap)->bounds.top)/2;
for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom)-1;j+=1)
        {
        hs = Img; //hold start
        ds = Dest;
        for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right)-1;i+=1)
                {
                double x,y;
                int tx,ty;
                double p1,p2,p3,p4;

                int r1,r2,r3,r4;
                int g1,g2,g3,g4;
                int b1,b2,b3,b4;
                int rtot,gtot,btot;
        //x=i;
        //y=j;
                x= (centerx + (( i -  centerx) * c_ang) - ((j - centery) * s_ang));
                y = (centery + (( j -  centery) * c_ang) + ((i - centerx) * s_ang));
        //      x = i * c_theta - j * s_theta;
        //      y = i * s_theta + j * c_theta;
                tx=x;//trunc
                ty=y;//trunc
```

FIG. 70

```
p1 = x - (double)tx;
p2 = 1.0 - p1;
p3 = y - (double)ty;
p4 = 1.0 - p3;

p1 /= 2; p2 /= 2; p3 /= 2; p4 /= 2;

if(ty < (*imap)->bounds.top)
        ;//*Img++;
else if(tx < (*imap)->bounds.left)
        ;
else if(tx >= centerx * 2)
        ;//*Img++;
else if(ty >= centery * 2)
        ;//*Img++;
else
        {

        r1 = RED(*(Img + (j*(ImgrowB/ROW_DIV)) + (i)));
        g1 = GREEN(*(Img + (j*(ImgrowB/ROW_DIV)) + (i)));
        b1 = BLUE(*(Img + (j*(ImgrowB/ROW_DIV)) + (i)));

        r2 = RED(*(Img + (j*(ImgrowB/ROW_DIV)) + (i+1)));
        g2 = GREEN(*(Img + (j*(ImgrowB/ROW_DIV)) + (i+1)));
        b2 = BLUE(*(Img + (j*(ImgrowB/ROW_DIV)) + (i+1)));

        r3 = RED(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i)));
        g3 = GREEN(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i)));
        b3 = BLUE(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i)));

        r4 = RED(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i+1)));
        g4 = GREEN(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i+1)));
        b4 = BLUE(*(Img + ((j+1)*(ImgrowB/ROW_DIV)) + (i+1)));

        rtot = r1 * p1 + r2 * p2 + r3 * p3 + r4 * p4;
        gtot = g1 * p1 + g2 * p2 + g3 * p3 + g4 * p4;
        btot = b1 * p1 + b2 * p2 + b3 * p3 + b4 * p4;

        *(Dest + (ty*(ImgrowB/ROW_DIV)) + (tx)) = ((rtot)<<16) | ((gtot)<<8) | (btot);//p1
        *(Dest + (ty*(ImgrowB/ROW_DIV)) + (tx+1)) = ((rtot)<<16) | ((gtot)<<8) | (btot);//p1
        }
}//for i

}
```

FIG. 71

```
//.....
Img = ImgStart;
Dest = DestStart;
if(inMask == 3)
        {
for(j=(*idestmap)->bounds.top+1;j< ((*idestmap)->bounds.bottom)-1;j+=1)
        {
        //hs = Img; //hold start
        //ds = Dest;
        for(i=(*idestmap)->bounds.left+1;i< ((*idestmap)->bounds.right)-1;i+=1)
                {
                int rt,gt,bt;
                int rl,gl,bl;
                int ra,ga,ba;
                int ri,gi,bi;
                int rr,gg,bb;

                ra = RED(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i-1)));
                ga = GREEN(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i-1)));
                ba = BLUE(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i-1)));

                rr = RED(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)));
                gg = GREEN(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)));
                bb = BLUE(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)));

                ri = RED(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i+1)));
                gi = GREEN(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i+1)));
                bi = BLUE(*(Dest + (j*(ImgrowB/ROW_DIV)) + (i+1)));

                rt = RED(*(Dest + (j-1*(ImgrowB/ROW_DIV)) + (i)));
                gt = GREEN(*(Dest + (j-1*(ImgrowB/ROW_DIV)) + (i)));
                bt = BLUE(*(Dest + (j-1*(ImgrowB/ROW_DIV)) + (i)));

                rl = RED(*(Dest + (j+1*(ImgrowB/ROW_DIV)) + (i)));
                gl = GREEN(*(Dest + (j+1*(ImgrowB/ROW_DIV)) + (i)));
                bl = BLUE(*(Dest + (j+1*(ImgrowB/ROW_DIV)) + (i)));

                *(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = (((rr*5+ri)/6)<<16) | (((gg*5+gi)/6)<<8) | (((bb*5+bi)/6));

                }//for i
        }
        }//inMask

//.....
```

FIG. 72

```
	HUnlock(imem);
	HUnlock((Handle)imap);
}

void MFlop(PixMapHandle idestmap,PixMapHandle imap,Handle imem,double angle)
{
	int i,j;
	int ImgrowB;
	long *Img,*ImgStart,*hs,*ds;
	long *Dest,*DestStart;
	int tx,ty;

	int row_w;
	int centerx,centery;


#define ROW_DIV  PIXEL_SIZE

	HLock(imem);
	HLock((Handle)imap);
	CalcBaseAddr(imap,imem);


	ImgrowB = (*imap)->rowBytes & 0x3fff;

	ImgStart = Img = (long *)(*imap)->baseAddr;
	DestStart = Dest = (long *)(*idestmap)->baseAddr;

	for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom);j+=1)
		for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right);i+=1)
			*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = 0;//fill blac to rid rot zones

	post("flop");

	row_w = (*imap)->bounds.right - (*imap)->bounds.left;

	centerx = ((*imap)->bounds.right - (*imap)->bounds.left)/2;
	centery = ((*imap)->bounds.bottom - (*imap)->bounds.top)/2;

	for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom);j+=1)
		{
		hs = Img; //hold start
		ds = Dest;
```

FIG. 73

```
                //tx = ((*imap)->bounds.right);
                //ty = j;
                for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right);i+=1)
                        {

                        tx=((*imap)->bounds.right) - (i-(*imap)->bounds.left);//norm
                        ty=j;//

                        *(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = (*(Img + (ty*(ImgrowB/ROW_DIV)) + (tx)));
                //      tx -= 1;
                        }//for i
                }
        //.....
        Img = ImgStart;
        Dest = DestStart;

        //.....
        HUnlock(imem);
        HUnlock((Handle)imap);
}

void MFlip(PixMapHandle idestmap,PixMapHandle imap,Handle imem,double angle)
{
        int i,j;
        int ImgrowB;
        long *Img,*ImgStart,*hs,*ds;
        long *Dest,*DestStart;
        int tx,ty;

        int row_w;
        int centerx,centery;


#define ROW_DIV  PIXEL_SIZE

        HLock(imem);
        HLock((Handle)imap);
        CalcBaseAddr(imap,imem);


        ImgrowB = (*imap)->rowBytes & 0x3fff;

        ImgStart = Img = (long *)(*imap)->baseAddr;
        DestStart = Dest = (long *)(*idestmap)->baseAddr;
```

FIG. 74

```
	for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom);j+=1)
		for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right);i+=1)
			*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = 0;//fill blac to rid rot zones

	post("flip");

	row_w = (*imap)->bounds.right - (*imap)->bounds.left;

	centerx = ((*imap)->bounds.right - (*imap)->bounds.left)/2;
	centery = ((*imap)->bounds.bottom - (*imap)->bounds.top)/2;

	for(j=(*imap)->bounds.top;j< ((*imap)->bounds.bottom);j+=1)
		{
		hs = Img; //hold start
		ds = Dest;

		//tx = ((*imap)->bounds.right);
		//ty = j;
		for(i=(*imap)->bounds.left;i< ((*imap)->bounds.right);i+=1)
			{

			tx=i;//norm
			ty=((*imap)->bounds.bottom) - (j-(*imap)->bounds.top)-1;//

			*(Dest + (j*(ImgrowB/ROW_DIV)) + (i)) = (*(Img + (ty*(ImgrowB/ROW_DIV)) + (tx)));
		//		tx -= 1;
			}//for i
		}
	//.....
	Img = ImgStart;
	Dest = DestStart;

	//.....
	HUnlock(imem);
	HUnlock((Handle)imap);
}
int Mabs(int x)
{
	if(x<0)
		return x * -1;
	else
		return x;
}
```

FIG. 75

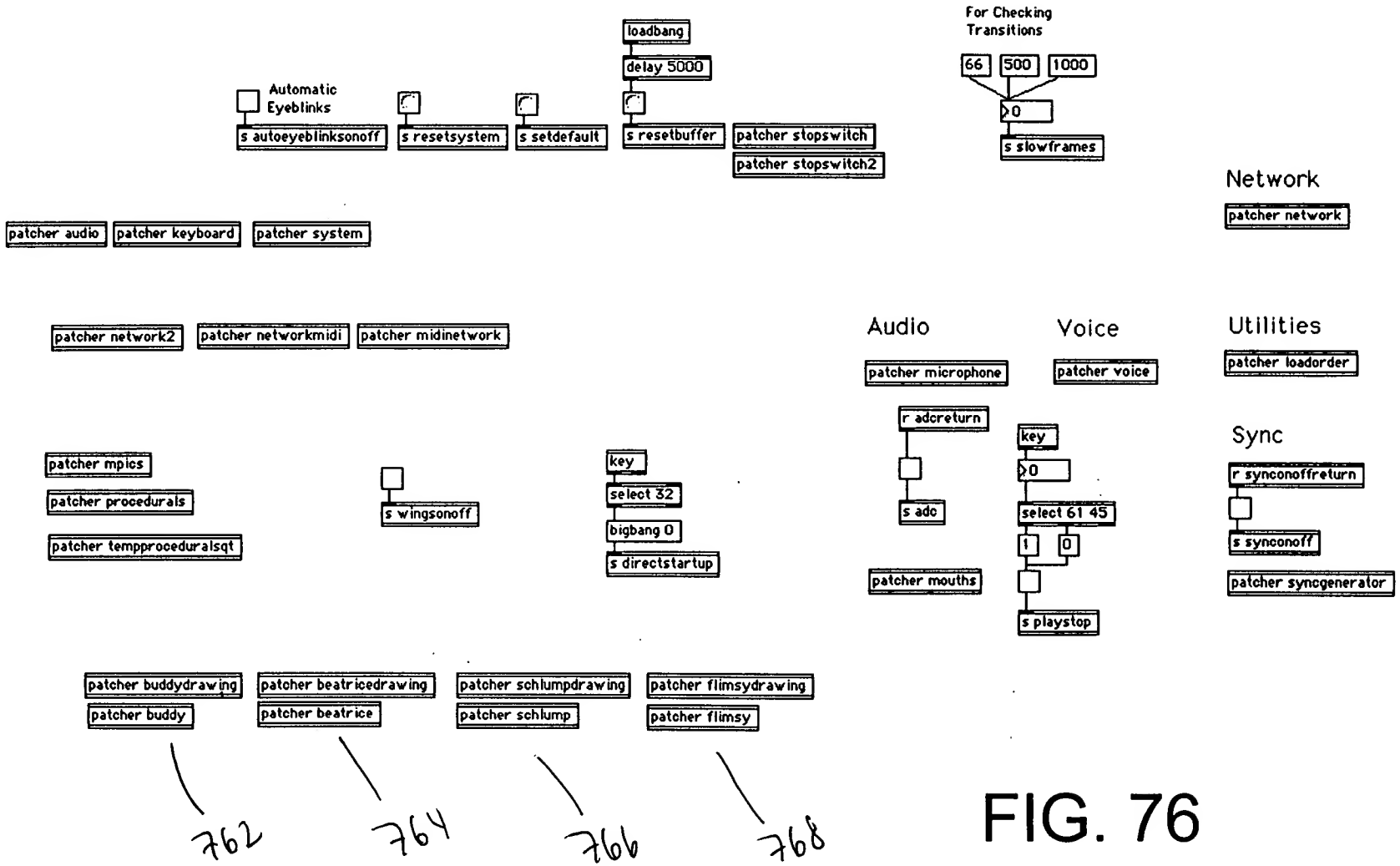

loadbang
delay 5000
Automatic Eyeblinks
s autoeyeblinksonoff
s resetsystem
s setdefault
s resetbuffer
patcher stopswitch
patcher stopswitch2
For Checking Transitions
66
500
1000
>0
s slowframes
Network
patcher network
patcher audio
patcher keyboard
patcher system
patcher network2
patcher networkmidi
patcher midinetwork
Audio
Voice
Utilities
patcher microphone
patcher voice
patcher loadorder
r adcreturn
key
Sync
patcher mpics
patcher procedurals
patcher tempproceduralsqt
s wingsonoff
key
select 32
bigbang 0
s directstartup
>0
s adc
select 61 45
r synconoffreturn
1
0
s synconoff
patcher mouths
s playstop
patcher syncgenerator
patcher buddydrawing
patcher buddy
patcher beatricedrawing
patcher beatrice
patcher schlumpdrawing
patcher schlump
patcher flimsydrawing
patcher flimsy
762
764
766
768

FIG. 76

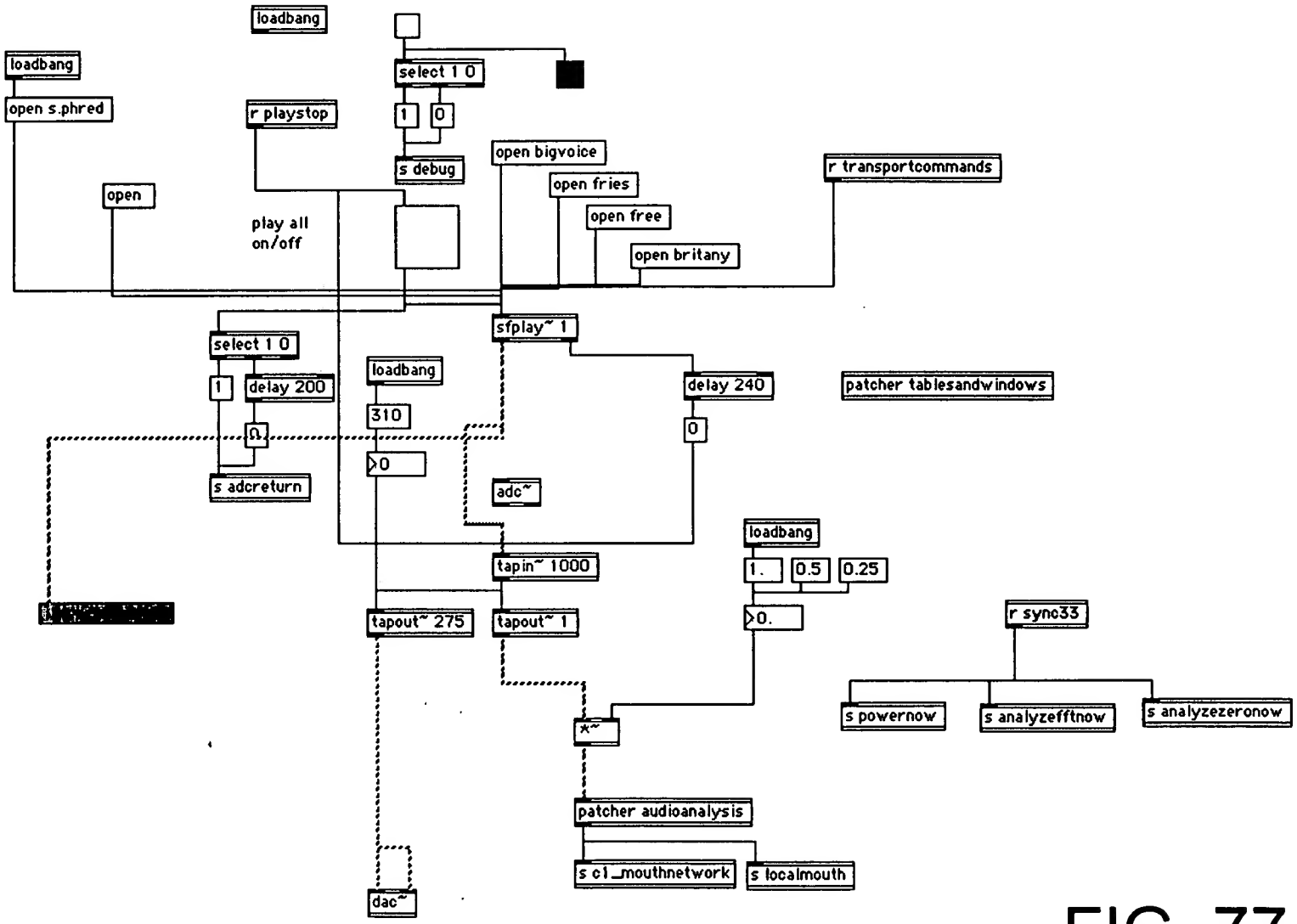

loadbang
loadbang
open s.phred
select 1 0
r playstop
1
0
s debug
open bigvoice
open fries
open free
open britany
r transportcommands
open
play all
on/off
sfplay~ 1
select 1 0
1
delay 200
loadbang
310
>0
s adcreturn
adc~
delay 240
0
patcher tablesandwindows
loadbang
1.
0.5
0.25
>0.
tapin~ 1000
tapout~ 275
tapout~ 1
r sync33
s powernow
s analyzefftnow
s analyzezeronow
*~
patcher audioanalysis
s cl_mouthnetwork
s localmouth
dac~

FIG. 77

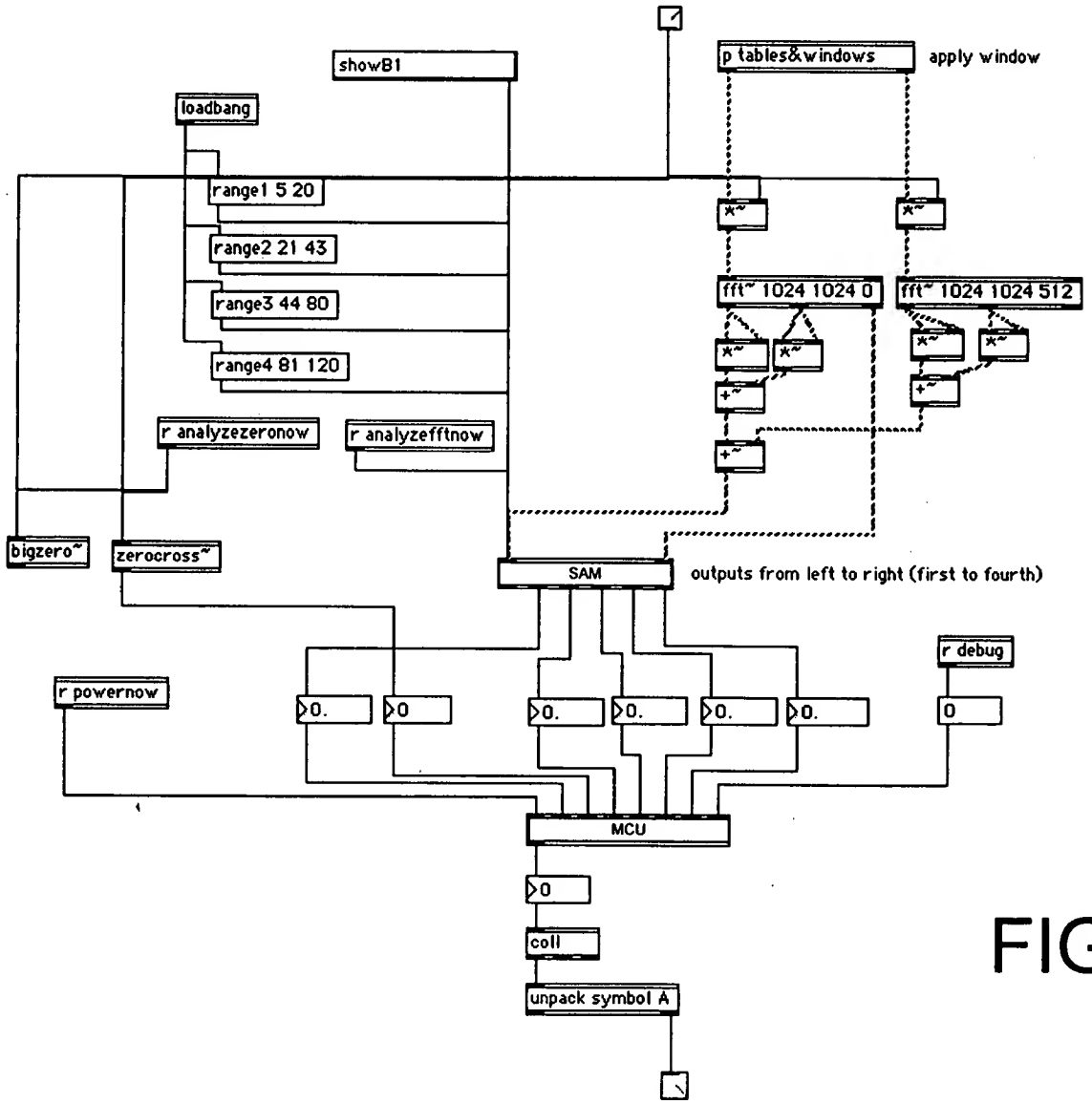

showB1
p tables&windows
apply window
loadbang
range1 5 20
range2 21 43
range3 44 80
range4 81 120
fft~ 1024 1024 0
fft~ 1024 1024 512
r analyzezeronow
r analyzefftnow
bigzero~
zerocross~
SAM
outputs from left to right (first to fourth)
r debug
r powernow
0.
0
0.
0.
0.
0.
0
MCU
0
coll
unpack symbol A

FIG. 78

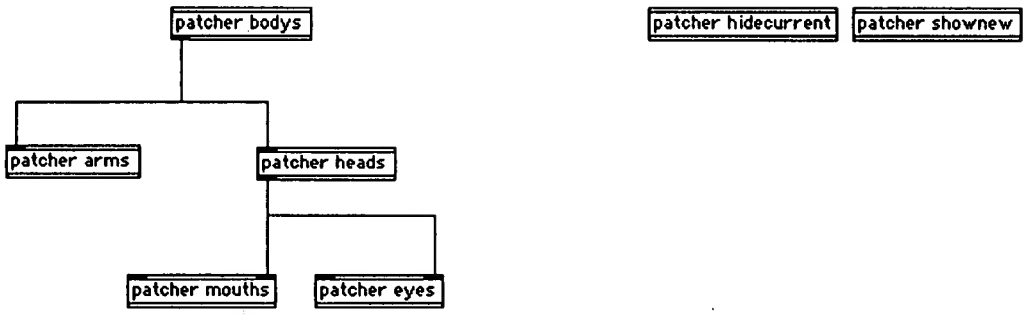

patcher bodys
patcher arms
patcher heads
patcher mouths
patcher eyes
patcher hidecurrent
patcher shownew

FIG. 79

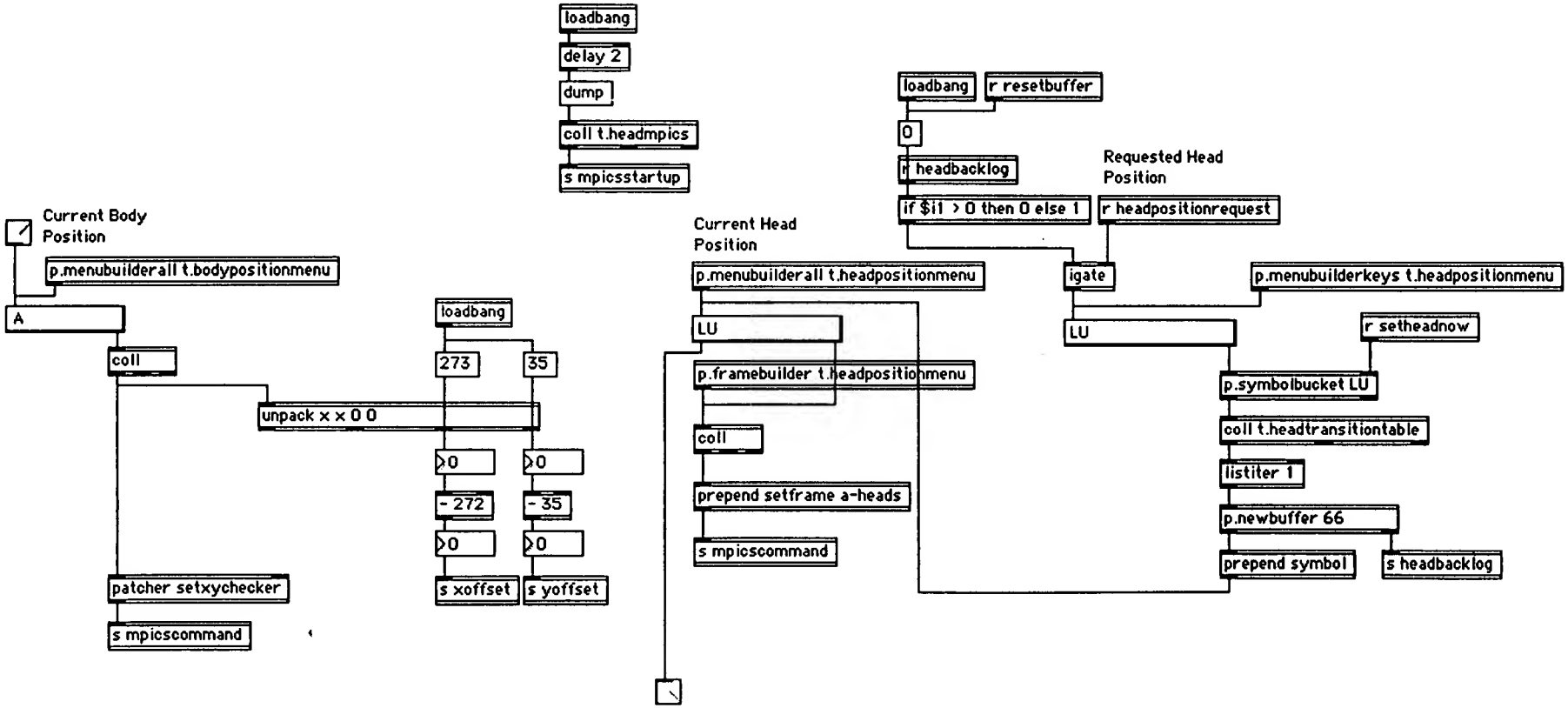
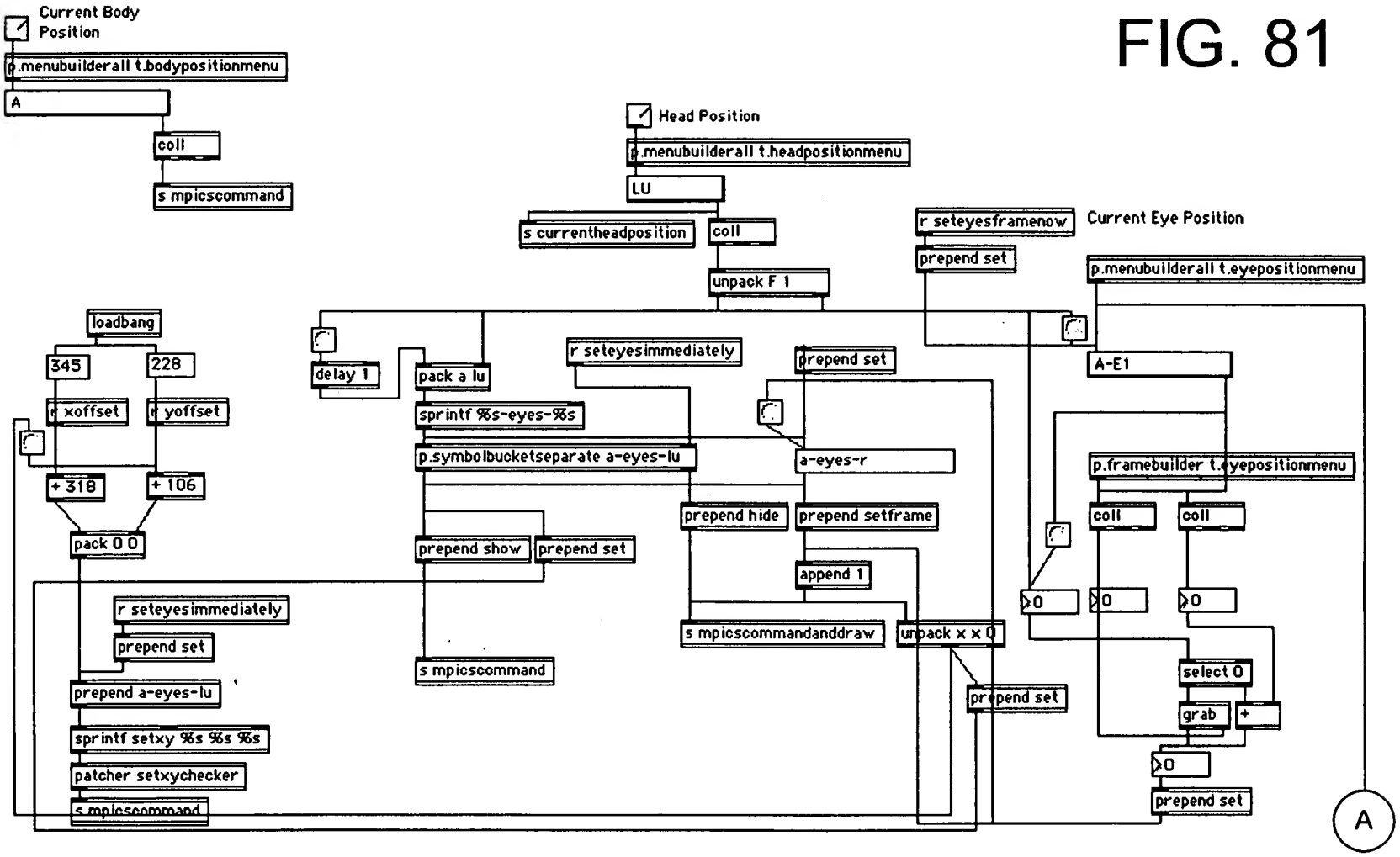

loadbang
delay 2
dump
coll t.headmpics
s mpicsstartup
Current Body Position
p.menubuilderall t.bodypositionmenu
A
coll
unpack x x 0 0
loadbang
273
35
>0
>0
- 272
- 35
>0
>0
s xoffset
s yoffset
patcher setxychecker
s mpicscommand
Current Head Position
p.menubuilderall t.headpositionmenu
LU
p.framebuilder t.headpositionmenu
coll
prepend setframe a-heads
s mpicscommand
loadbang
r resetbuffer
0
r headbacklog
if $i1 > 0 then 0 else 1
Requested Head Position
r headpositionrequest
igate
p.menubuilderkeys t.headpositionmenu
LU
r setheadnow
p.symbolbucket LU
coll t.headtransitiontable
listiter 1
p.newbuffer 66
prepend symbol
s headbacklog

FIG. 80

FIG. 81


Current Body Position
p.menubuilderall t.bodypositionmenu
A
coll
s mpicscommand
Head Position
p.menubuilderall t.headpositionmenu
LU
s currentheadposition
coll
unpack F 1
r seteyesframenow
Current Eye Position
prepend set
p.menubuilderall t.eyepositionmenu
loadbang
345
228
r xoffset
r yoffset
+ 318
+ 106
pack 0 0
delay 1
pack a lu
r seteyesimmediately
prepend set
A-E1
sprintf %s-eyes-%s
p.symbolbucketseparate a-eyes-lu
a-eyes-r
p.framebuilder t.eyepositionmenu
prepend hide
prepend setframe
coll
coll
prepend show
prepend set
append 1
>0
>0
>0
r seteyesimmediately
prepend set
s mpicscommandanddraw
unpack x x 0
s mpicscommand
select 0
grab
+
prepend a-eyes-lu
prepend set
sprintf setxy %s %s %s
patcher setxychecker
s mpicscommand
>0
prepend set
A

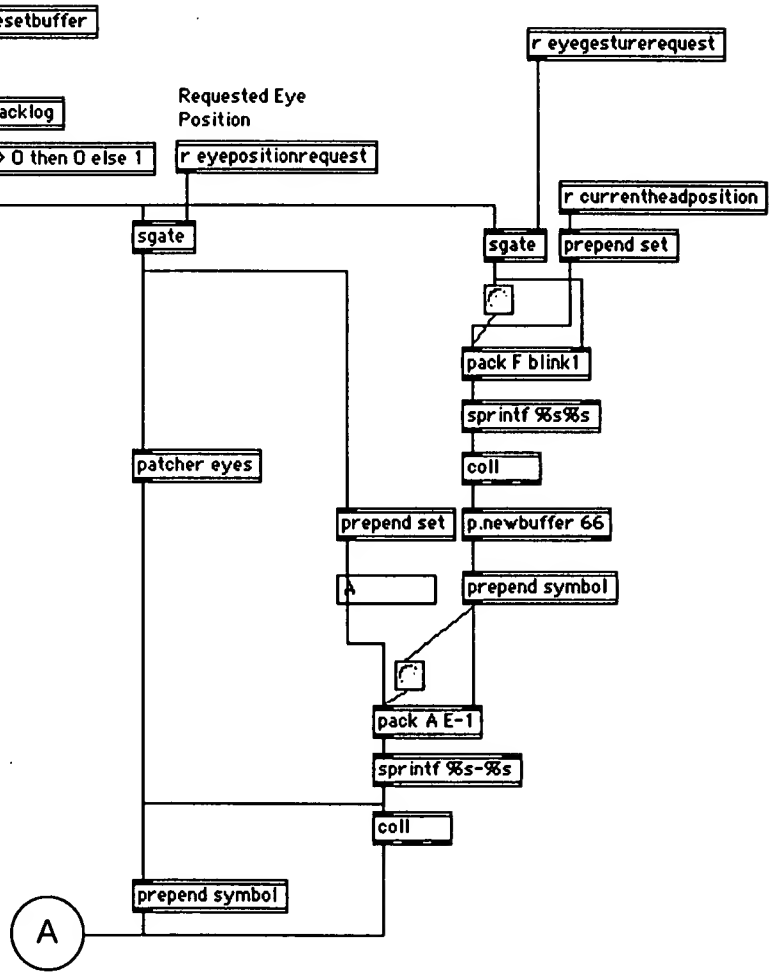

loadbang
r resetbuffer
0
r eyebacklog
if $i1 > 0 then 0 else 1
Requested Eye Position
r eyepositionrequest
r eyegesturerequest
r currentheadposition
sgate
sgate
prepend set
pack F blink1
sprintf %s%s
coll
patcher eyes
prepend set
p newbuffer 66
A
prepend symbol
pack A E-1
sprintf %s-%s
coll
prepend symbol
A

FIG. 82

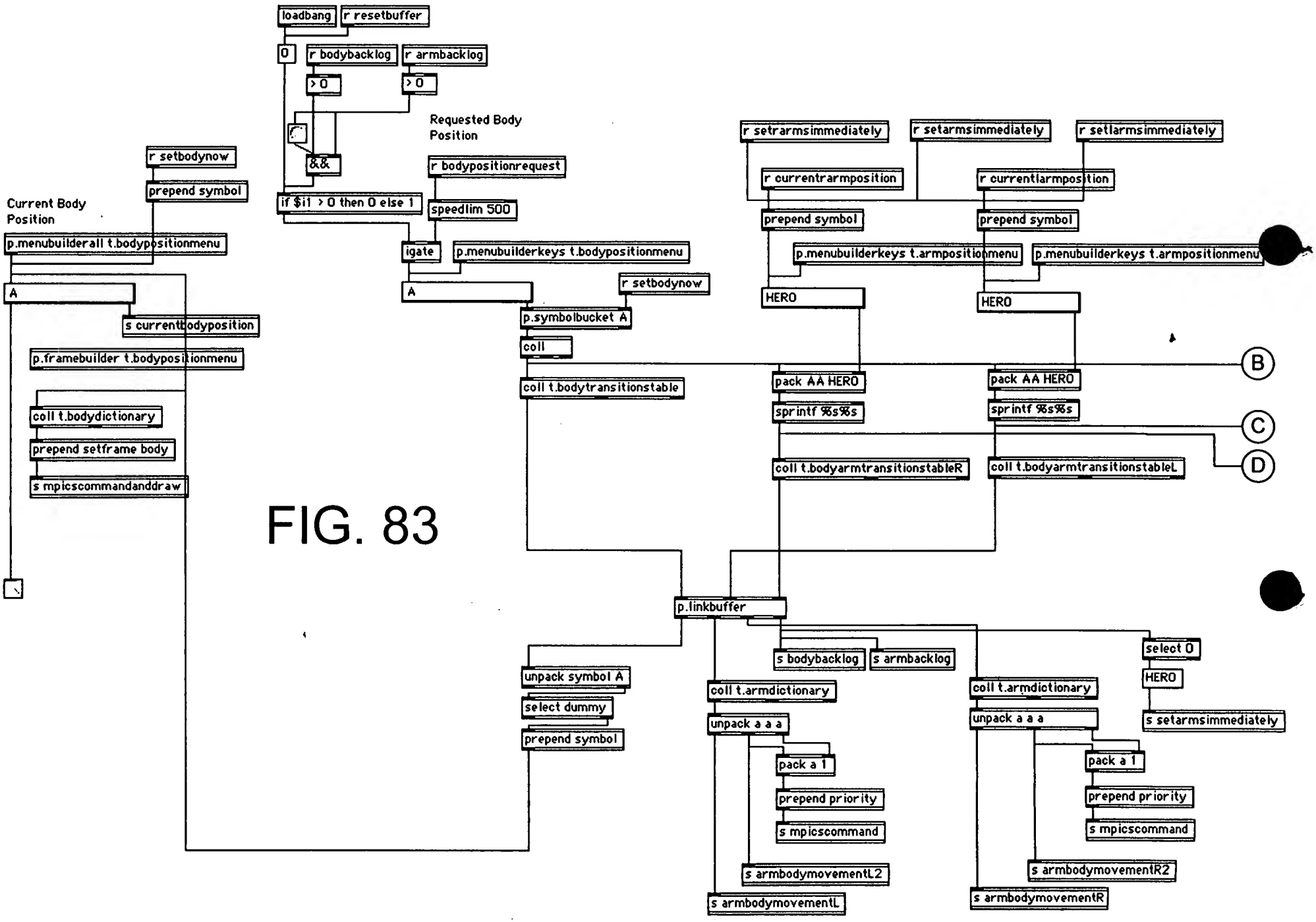
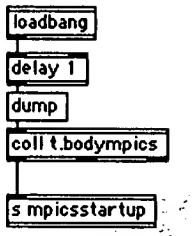

loadbang
r resetbuffer
0
r bodybacklog
r armbacklog
> 0
> 0
&&
if $i1 > 0 then 0 else 1
Requested Body Position
r bodypositionrequest
speedlim 500
igate
p.menubuilderkeys t.bodypositionmenu
A
r setbodynow
p.symbolbucket A
coll
coll t.bodytransitionstable
r setbodynow
prepend symbol
Current Body Position
p.menubuilderall t.bodypositionmenu
A
s currentbodyposition
p.framebuilder t.bodypositionmenu
coll t.bodydictionary
prepend setframe body
s mpicscommandanddraw
r setrarmsimmediately
r setarmsimmediately
r setlarmsimmediately
r currentrarmposition
r currentlarmposition
prepend symbol
prepend symbol
p.menubuilderkeys t.armpositionmenu
p.menubuilderkeys t.armpositionmenu
HERO
HERO
B
pack AA HERO
pack AA HERO
sprintf %s%s
sprintf %s%s
C
D
coll t.bodyarmtransitionstableR
coll t.bodyarmtransitionstableL
p.linkbuffer
s bodybacklog
s armbacklog
unpack symbol A
select dummy
prepend symbol
coll t.armdictionary
unpack a a a
pack a 1
prepend priority
s mpicscommand
s armbodymovementL2
s armbodymovementL
coll t.armdictionary
unpack a a a
pack a 1
prepend priority
s mpicscommand
s armbodymovementR2
s armbodymovementR
select 0
HERO
s setarmsimmediately

FIG. 83

FIG. 84


loadbang
delay 1
dump
coll t.bodympics
s mpicsstartup

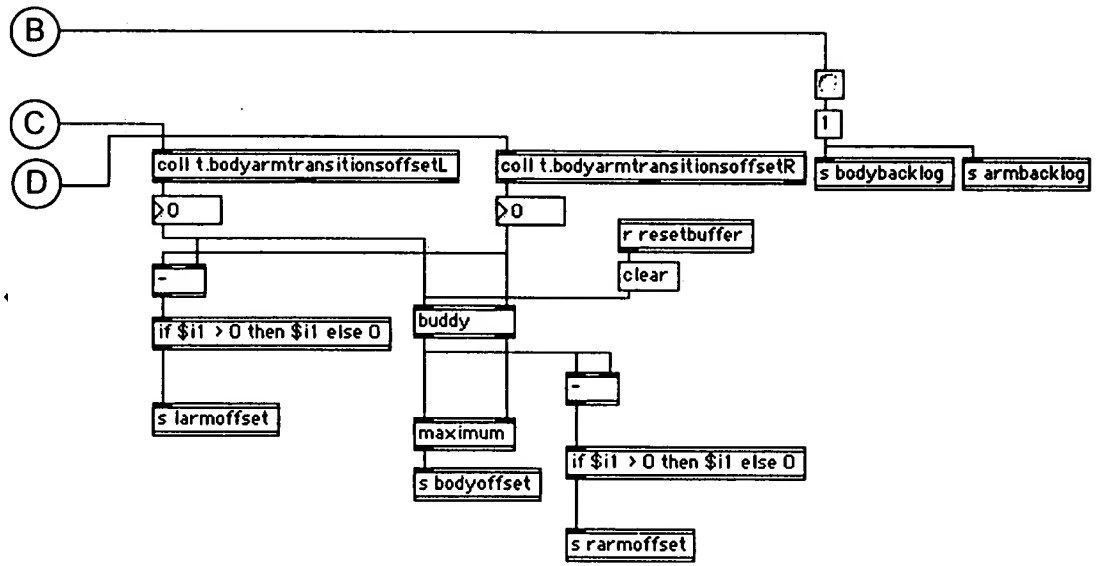
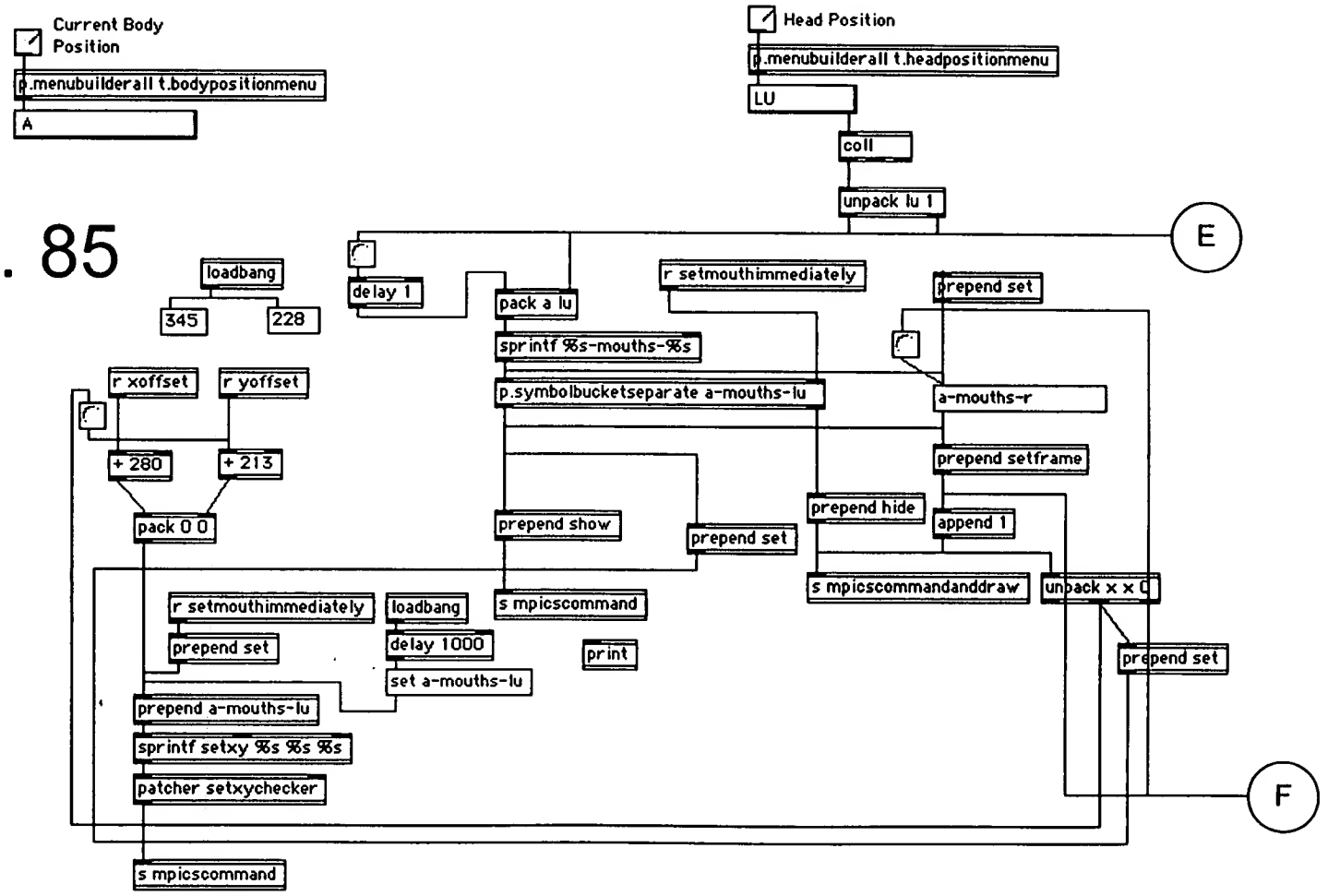

B
C
D
coll t.bodyarmtransitionsoffsetL
coll t.bodyarmtransitionsoffsetR
s bodybacklog
s armbacklog
>0
>0
r resetbuffer
clear
-
buddy
if $i1 > 0 then $i1 else 0
-
s larmoffset
maximum
if $i1 > 0 then $i1 else 0
s bodyoffset
s rarmoffset

FIG. 85


Current Body Position
p.menubuilderall t.bodypositionmenu
A
Head Position
p.menubuilderall t.headpositionmenu
LU
coll
unpack lu 1
E
loadbang
345
228
delay 1
pack a lu
r setmouthimmediately
prepend set
sprintf %s-mouths-%s
p.symbolbucketseparate a-mouths-lu
a-mouths-r
r xoffset
r yoffset
+ 280
+ 213
pack 0 0
prepend setframe
prepend hide
append 1
prepend show
prepend set
s mpicscommandanddraw
unpack x x 0
r setmouthimmediately
loadbang
s mpicscommand
prepend set
delay 1000
print
prepend set
set a-mouths-lu
prepend a-mouths-lu
sprintf setxy %s %s %s
patcher setxychecker
F
s mpicscommand

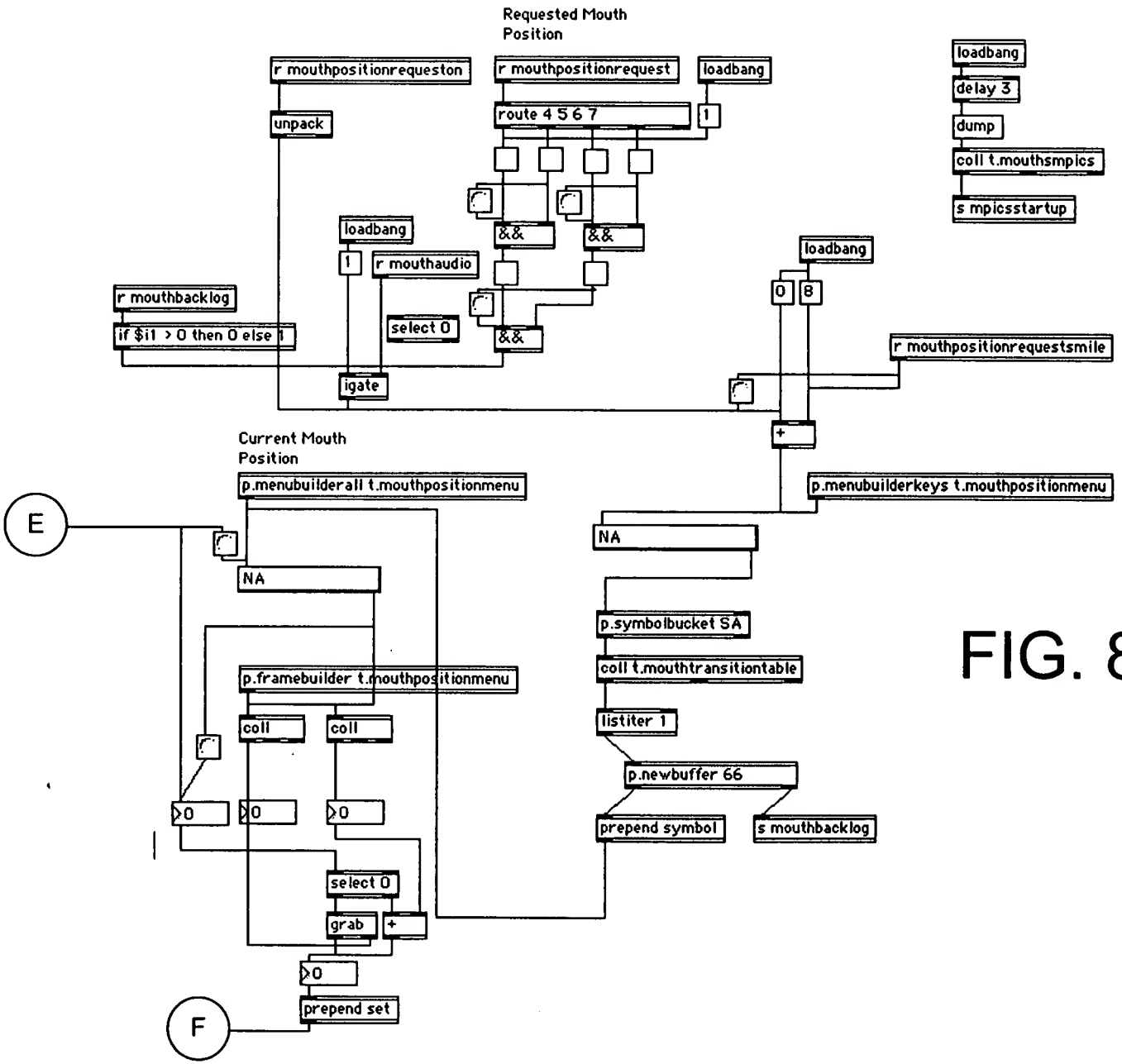

Requested Mouth Position
r mouthpositionrequeston
r mouthpositionrequest
loadbang
unpack
route 4 5 6 7
1
loadbang
delay 3
dump
coll t.mouthsmpics
s mpicsstartup
loadbang
1
r mouthaudio
&&
&&
r mouthbacklog
if $i1 > 0 then 0 else 1
select 0
&&
igate
loadbang
0
8
r mouthpositionrequestsmile
+
Current Mouth Position
p.menubuilderall t.mouthpositionmenu
p.menubuilderkeys t.mouthpositionmenu
E
NA
NA
p.symbolbucket SA
p.framebuilder t.mouthpositionmenu
coll t.mouthtransitiontable
coll
coll
listiter 1
p.newbuffer 66
>0
>0
>0
prepend symbol
s mouthbacklog
select 0
grab
+
>0
prepend set
F

FIG. 86

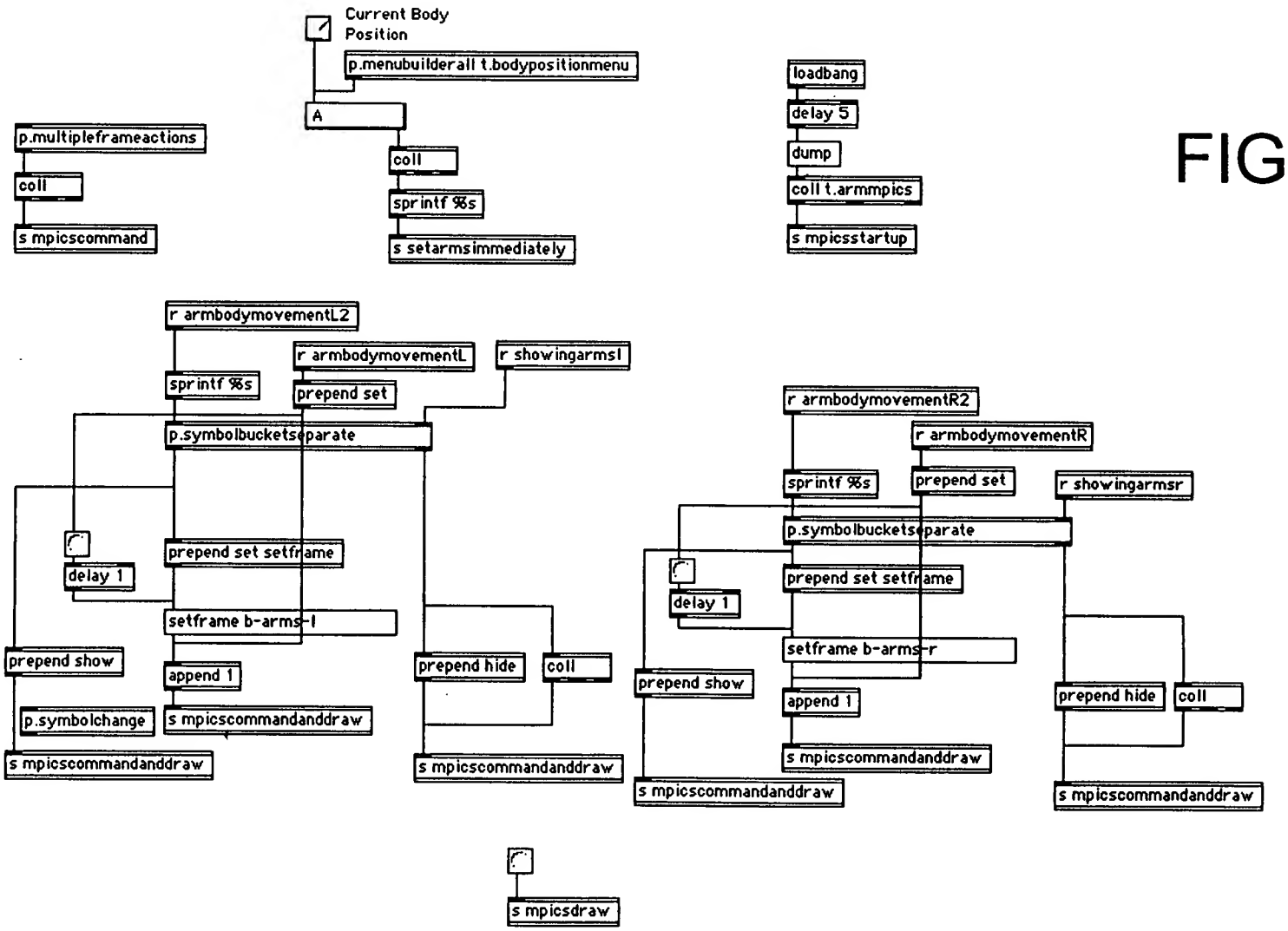

Current Body Position
p.menubuilderall t.bodypositionmenu
A
coll
sprintf %s
s setarmsimmediately
p.multipleframeactions
coll
s mpicscommand
loadbang
delay 5
dump
coll t.armmpics
s mpicsstartup
r armbodymovementL2
r armbodymovementL
r showingarmsl
sprintf %s
prepend set
p.symbolbucketseparate
delay 1
prepend set setframe
setframe b-arms-l
prepend show
append 1
p.symbolchange
s mpicscommandanddraw
s mpicscommandanddraw
prepend hide
coll
s mpicscommandanddraw
r armbodymovementR2
r armbodymovementR
sprintf %s
prepend set
r showingarmsr
p.symbolbucketseparate
delay 1
prepend set setframe
setframe b-arms-r
prepend show
append 1
s mpicscommandanddraw
s mpicscommandanddraw
prepend hide
coll
s mpicscommandanddraw
s mpicsdraw

FIG. 87

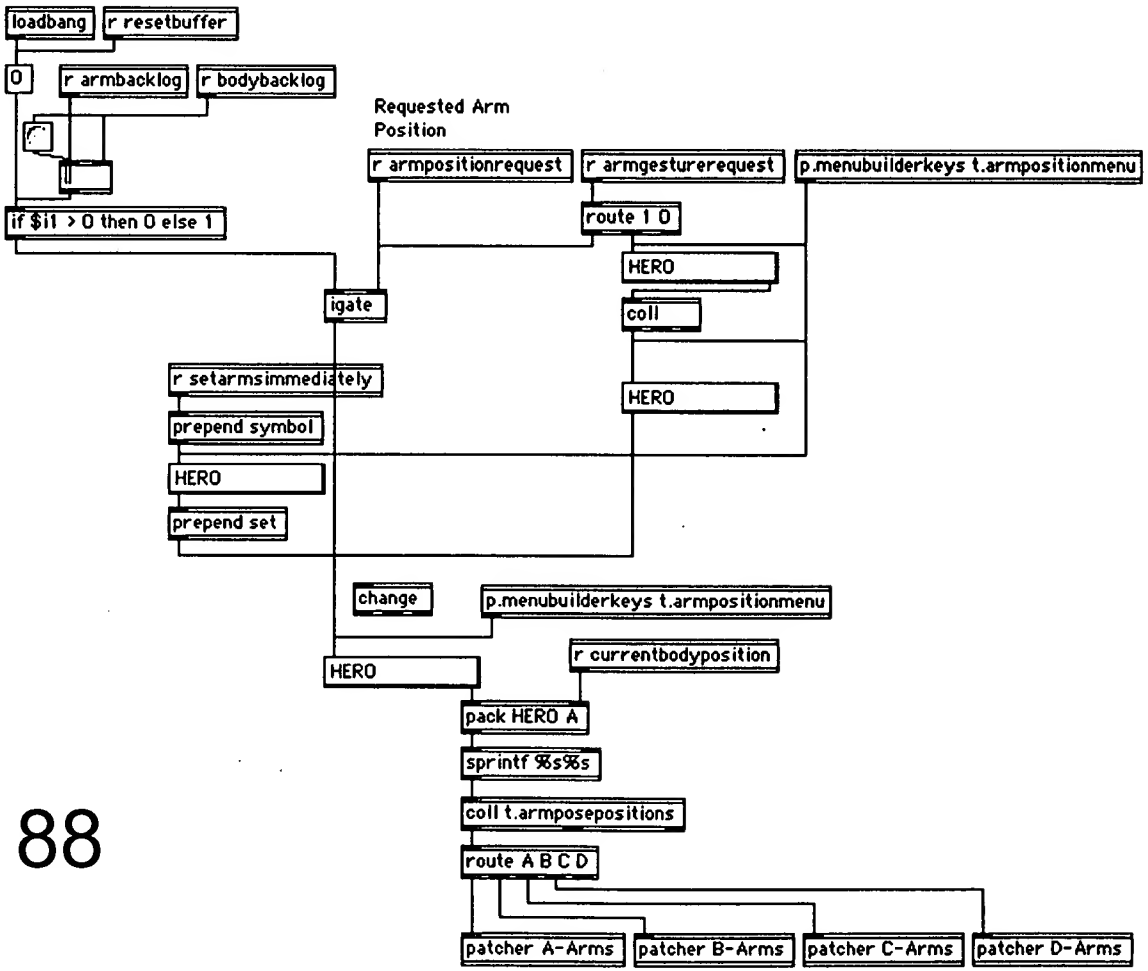

loadbang
r resetbuffer
0
r armbacklog
r bodybacklog
if $i1 > 0 then 0 else 1
Requested Arm Position
r armpositionrequest
r armgesturerequest
p.menubuilderkeys t.armpositionmenu
route 1 0
HERO
coll
HERO
igate
r setarmsimmediately
prepend symbol
HERO
prepend set
change
p.menubuilderkeys t.armpositionmenu
r currentbodyposition
HERO
pack HERO A
sprintf %s%s
coll t.armposepositions
route A B C D
patcher A-Arms
patcher B-Arms
patcher C-Arms
patcher D-Arms

FIG. 88

FIG. 89